THE CIP NETWORKS LIBRARY

Volume 7

Integration of Modbus Devices into the CIP Architecture

Edition 1.0

November 2007

ODVA & ControlNet International, Ltd.

The CIP Networks Library
Volume 7: Modbus Integration into the CIP Architecture

Publication Number: PUB00202

Open DeviceNet Vendor Association, Inc.
4220 Varsity Drive, Suite Ann Arbor, MI 48108-5006 USA
TEL 1-734-975-8840
FAX 1-734-922-0027
EMAIL odva@odva.org
WEB www.odva.org

The CIP Networks Library: Volume 7

Integration of Modbus Devices into the CIP Architecture

Table of Contents

Revisions

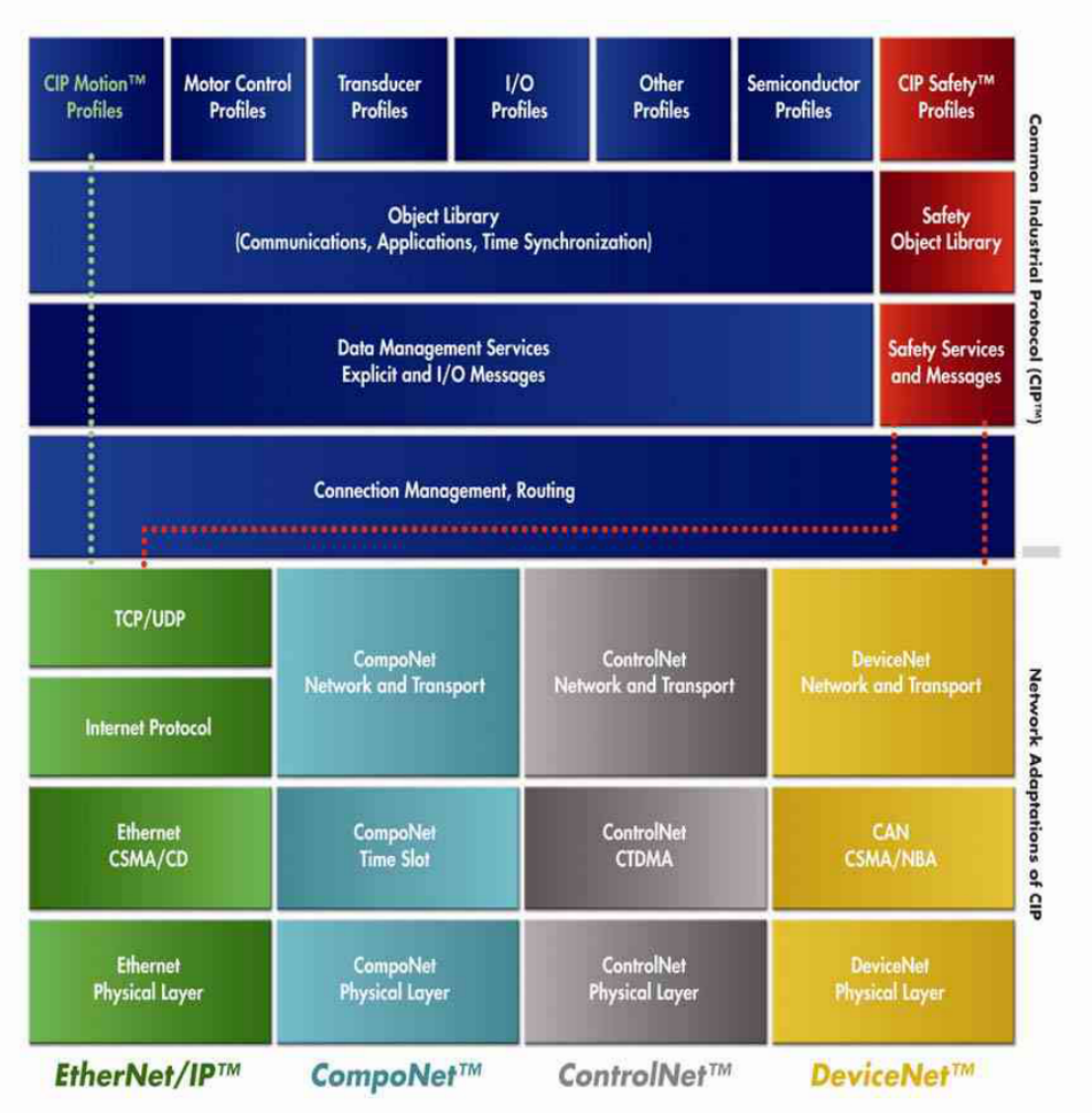
This is the first release of this volume.

Preface

Organization of the CIP Networks Specifications

Today, four networks - DeviceNet™, ControlNet™, EtherNet/IP™ and CompoNet™ - use the Common Industrial Protocol (CIP) for the upper layers of their network protocol. For this reason, the associations that manage these networks - ODVA and ControlNet International - have mutually agreed to manage and distribute the specifications for CIP Networks in a common structure to help ensure consistency and accuracy in the management of these specifications.

The following diagram illustrates the organization of the library of CIP Network specifications. In addition to CIP Networks, CIP Safety™ consists of the extensions to CIP for functional safety.



This common structure presents CIP in one volume with a separate volume for each network adaptation of CIP. The specifications for the CIP Networks are two-volume sets, paired as shown below.

The EtherNet/IP specification consists of:

Volume 1: Common Industrial Protocol (CIP™)

Volume 2: EtherNet/IP Adaptation of CIP

The DeviceNet specification consists of:

Volume 1: Common Industrial Protocol (CIP™)

Volume 3: DeviceNet Adaptation of CIP

The ControlNet specification consists of:

Volume 1: Common Industrial Protocol (CIP™)

Volume 4: ControlNet Adaptation of CIP

The CompoNet specification consists of:

Volume 1: Common Industrial Protocol (CIP™)

Volume 6: CompoNet Adaptation of CIP

The specification for CIP Safety™ is distributed in a single volume:

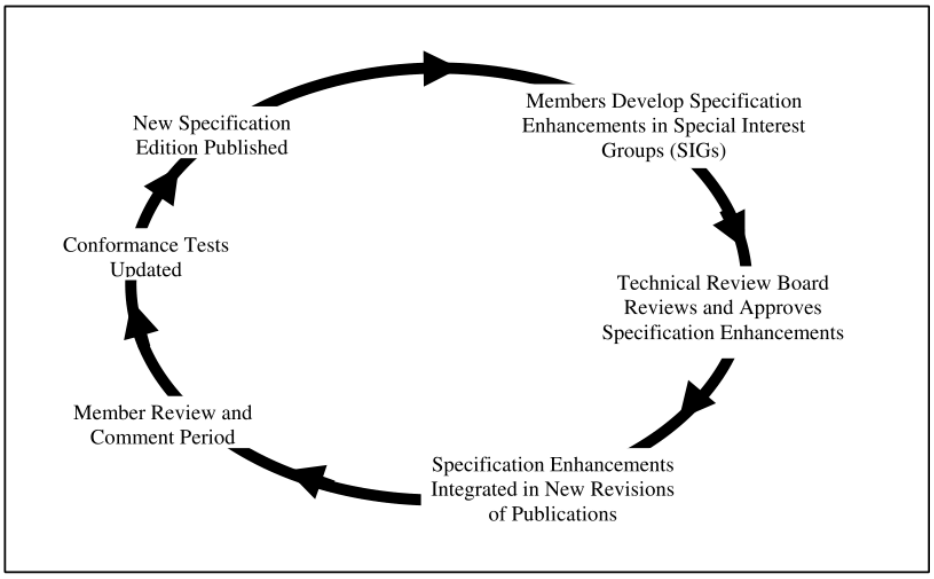
Volume 5: CIP Safety

The specification for integrating Modbus Devices is distributed in a single volume:

Volume 7: Integration of Modbus Devices into the CIP Architecture

Specification Enhancement Process

The specifications for CIP Networks are continually being enhanced to meet the increasing needs of users for features and functionality. ODVA and ControlNet International have also agreed to operate using a common Specification Enhancement Process in order to ensure open and stable specifications for all CIP Networks. This process is on going throughout the year for each CIP Network Specification as shown in the figure below. New editions of each CIP Network specification are published on a periodic basis.



Volume 7: Integration of Modbus Devices into the CIP Architecture

Chapter 1: Introduction

Contents

1-1 Introduction

This volume defines a standard mechanism for integrating Modbus server devices into the CIP protocol suite. With this mechanism, a CIP originator can communicate with a Modbus device as if it is a native CIP device. From the CIP originator's perspective, the Modbus device appears as a CIP target device, with CIP objects and with CIP communication mechanisms. From the Modbus device perspective, the originating CIP device appears as a Modbus client.

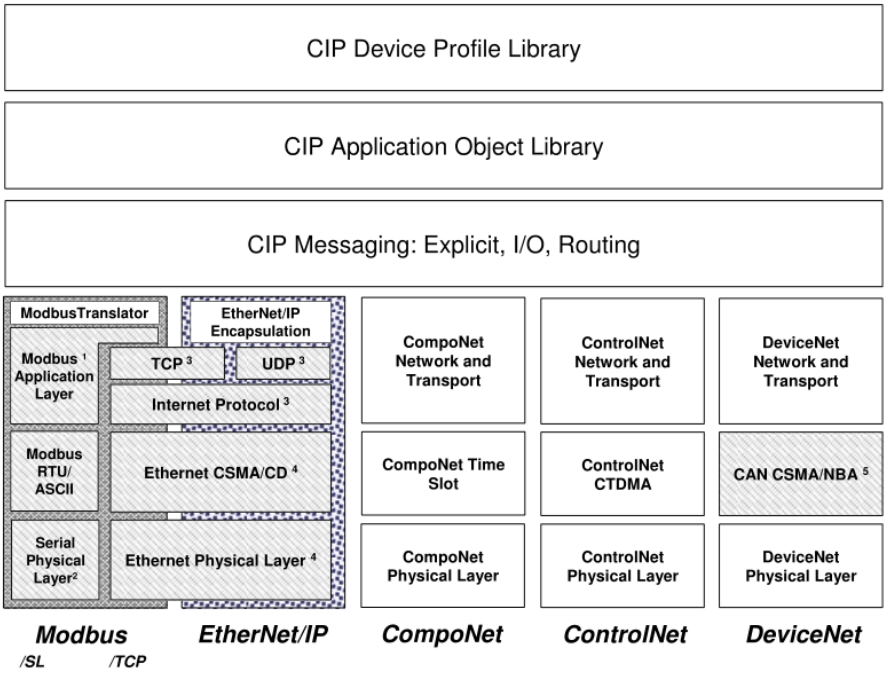
The Modbus integration mechanism includes the following components:

- The Modbus Translator that translates CIP objects and services to Modbus/TCP or Modbus serial messages. The Translator can be implemented in a CIP originator or in a CIP router.
- A defined set of CIP objects and services that allow basic read and write access to Modbus data. Support is also provided for execution of any Modbus function code via the Modbus Object.
- Definition of explicit and implicit CIP communications from the CIP originator to the Modbus target device. The Translator converts CIP explicit and implicit messages to Modbus requests, and converts Modbus responses to CIP messages.
- EDS files that describe the Modbus device capabilities in CIP terms. A generic Modbus EDS file is defined for use with any Modbus device. A device-specific EDS file may optionally be defined.

The result is seamless communication between CIP-based devices and Modbus-based devices, without requiring changes to existing Modbus devices or CIP originators.

Figure 1-1.1 shows Modbus Device integration in the CIP network architecture.

Figure 1-1.1 Modbus Device Integration in the CIP Network Architecture



Footnotes for Figure 1-1.1:

1. Modbus is defined by the Modbus specification, which is managed by Modbus-IDA. (http://www.modbus-ida.org)
2. Modbus RTU/ASCII uses EIA/TIA-232, TIA-422 or EIA/TIA-485 standards for serial communications. (EIA) is the Electronics Industries Alliance (http//www.eia.org) and (TIA) is the Telecommunications Industry Association (http://tiaonline.org).
3. TCP (RFC 793), UDP (RFC 768) and the Internet Protocol (RFC 791) are managed by The Internet Engineering Task Force (IETF). (http://www.ietf.org)
4. Ethernet CSMA/CD and Ethernet Physical Layer are defined by IEEE 802.3, which is managed by the Institute of Electrical and Electronic Engineers. (http://www.ieee.org)
5. CAN CSMA/NBA is ISO 11898, defined by the International Organization for Standardization. (http://www.iso.org)

1-2 Who Should Read This Volume?

The information in this volume is primarily intended for the following users:

- Developers of CIP originator devices who wish to implement the Modbus Translator in their device, or wish to allow their devices to communicate via a CIP-to-Modbus router.
- Developers of CIP router devices who wish to implement the Modbus Translator in a CIP-to-Modbus router (e.g., an EtherNet/IP to Modbus Serial router).
- Modbus device vendors who wish to understand how their device can be integrated into the CIP to Modbus integration solution (e.g., to be able to create EDS files for Modbus devices).

1-3 Scope

This specification is meant to be used in conjunction with the existing volumes of the CIP Networks Library – Volume 1: Common Industrial Protocol (CIP), Volume 2: EtherNet/IP Adaptation of CIP, Volume 3: DeviceNet Adaptation of CIP, Volume 4: ControlNet Adaptation of CIP and Volume 6: CompoNet Adaptation of CIP.

A CIP device that implements the Modbus Translator will additionally adhere to its corresponding CIP network volume. For example, an EtherNet/IP device that includes the Modbus Translator will follow the EtherNet/IP volume in addition to this volume.

Modbus devices are not intended to require changes as a result of the CIP to Modbus integration mechanism. This specification identifies the assumptions, recommendations, and optional features for Modbus devices to fully participate in the CIP to Modbus integration solution.

This specification is divided into the following chapters. (Some chapters may not have content)

Chapter	Title	Description
1	Introduction	This chapter of the specification.
2	Overview of Modbus Device Integration	Contains an overview of the Modbus Device integration solution.
3	Integrating Modbus Devices with CIP	Defines recommendations and options for Modbus devices in order to integrate with the CIP to Modbus integration solution.
4	Object Model	CIP to Modbus specific additions to the standard object model.
5	Object Library	Supplements the CIP object library with objects specific to Modbus.
6	Device Profiles	Contains Modbus-specific additions to the CIP device profile library.
7	Electronic Data Sheets	Specifies additions to the CIP EDS definition required for Modbus devices.
8	Physical Layer	CIP to Modbus specific additions to the physical layer.
9	Indicators and Middle Layers	CIP to Modbus specific additions to the indicators and middle layers.
10	Modbus Translator	Defines the Modbus Translator, which may be implemented in a CIP originator or CIP router.

1-4 References

Modbus Application Protocol Specification, V1.1b (www.modbus-ida.org)

Modbus Messaging on TCP/IP Implementation Guide, V1.0b (www.modbus-ida.org)

Modbus over Serial Line Specification and Implementation Guide, V1.02 (www.modbus-ida.org)

1-5 Definitions

For the purposes of this standard, the following definitions apply. Also see the Volume 1, Chapter 1 for additional definitions.

Term	Definition
EtherNet/IP	The implementation of the Common Industrial Protocol over the TCP/IP protocol suite and Ethernet (and Ethernet-like) physical media. The "IP" in EtherNet/IP stands for "Industrial Protocol" and should not be confused with the "Internet Protocol" that is part of the TCP/IP protocol suite. EtherNet/IP is defined by Volume 1 and Volume 2 of the CIP Networks Specification.
Modbus	Modbus is an application layer messaging protocol, positioned at layer 7 of the OSI model that provides client/server communication between devices connected on different buses or networks.
Modbus/TCP	The Modbus application protocol transported using the TCP/IP protocol suite.
Modbus Serial (also written as Modbus SL)	The Modbus application protocol transported over serial networks TIA/EIA-485 and TIA/EIA-232. Includes both RTU and ASCII modes.

1-6 Abbreviations

For the purposes of this standard, the following abbreviations apply. Also see the Volume 1, Chapter 1 for additional abbreviations.

Abbreviation	Meaning
MEI	Modbus Encapsulation Interface. For more information, see Modbus Application Protocol Specification reference in section 1-4.

Volume 7: Integration of Modbus Devices into the CIP Architecture

Chapter 2: Overview of Modbus Device Integration

Contents

2-1 Introduction

This chapter presents an overview of how Modbus devices are integrated with the CIP network architecture. Refer to the subsequent chapters for more details of the Modbus device integration mechanisms.

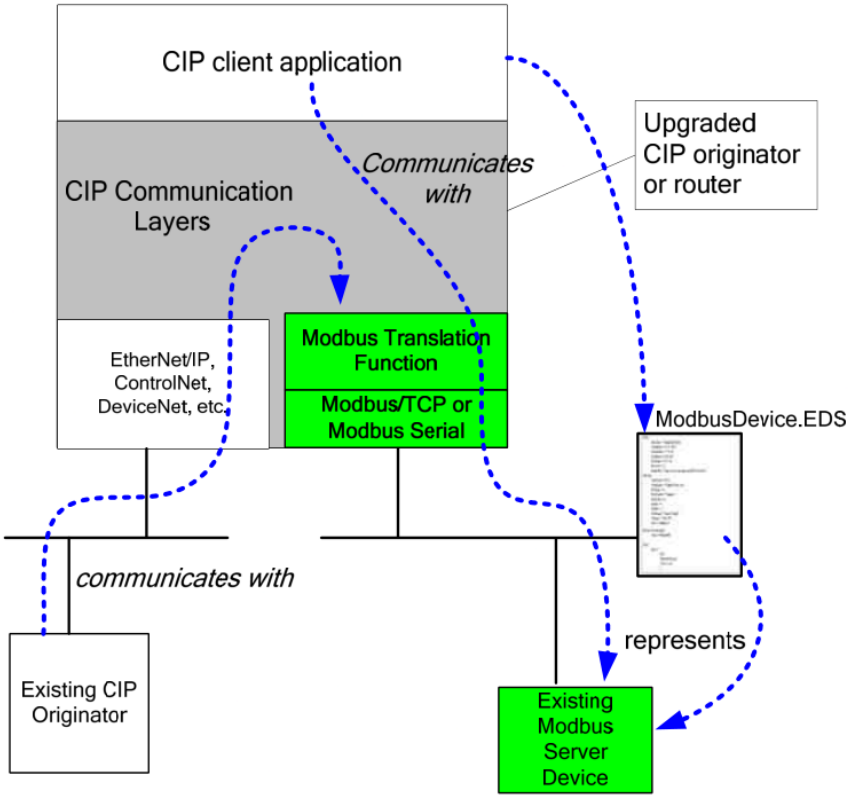
2-2 Model for Modbus Device Integration with CIP Networks

The overall goal of Modbus device integration with CIP is to provide seamless communications between CIP originator and Modbus server devices. Since the installed base of Modbus devices is large, a primary constraint is not to require modification to the Modbus server device in order to support CIP communications. It is also desirable to shield CIP applications from the details of the Modbus protocol itself.

In the resulting approach, the Modbus server device appears to the CIP originator as if it is a CIP device. The CIP originator is able to use familiar CIP object and communication mechanisms to communicate with the Modbus server device.

A Translator within the CIP originator's CIP stack, or in a CIP router device, performs the translation between CIP and Modbus. The solution provides for both explicit and implicit communication from the CIP originator to the Modbus Device.

Figure 2-2.1 Model for Modbus Device Integration within CIP



The following list summarizes the essential elements of the Modbus device integration solution:

- Modbus devices require no changes in order to support the Modbus device integration mechanism. It is recommended that the Modbus devices support Modbus function codes as stated in later chapters.
- To the CIP originator, the Modbus server device appears as a CIP device. The Modbus device has a device profile (with a specific device type), with a defined set of CIP objects and services that allow access to Modbus data tables.
- The Modbus Translator translates CIP objects and services to Modbus/TCP or Modbus serial messages. The Translator can be implemented in a CIP originator or in a CIP router.
- CIP objects and services allow basic read and write access to Modbus data. Support is also provided for direct execution of any Modbus function code via a new CIP Modbus Object.
- Both explicit and implicit CIP communications are supported from the CIP originator to the Modbus target device. The Translator converts CIP explicit and implicit messages to Modbus requests, and converts Modbus responses to CIP messages.
- EDS files are supported to describe the Modbus device capabilities in CIP terms. A generic Modbus EDS file is defined for use with any Modbus device. A device-specific EDS file may optionally be defined by the device vendor.

2-3 Modbus Translator Solutions

2-3.1 Translator Solution Options

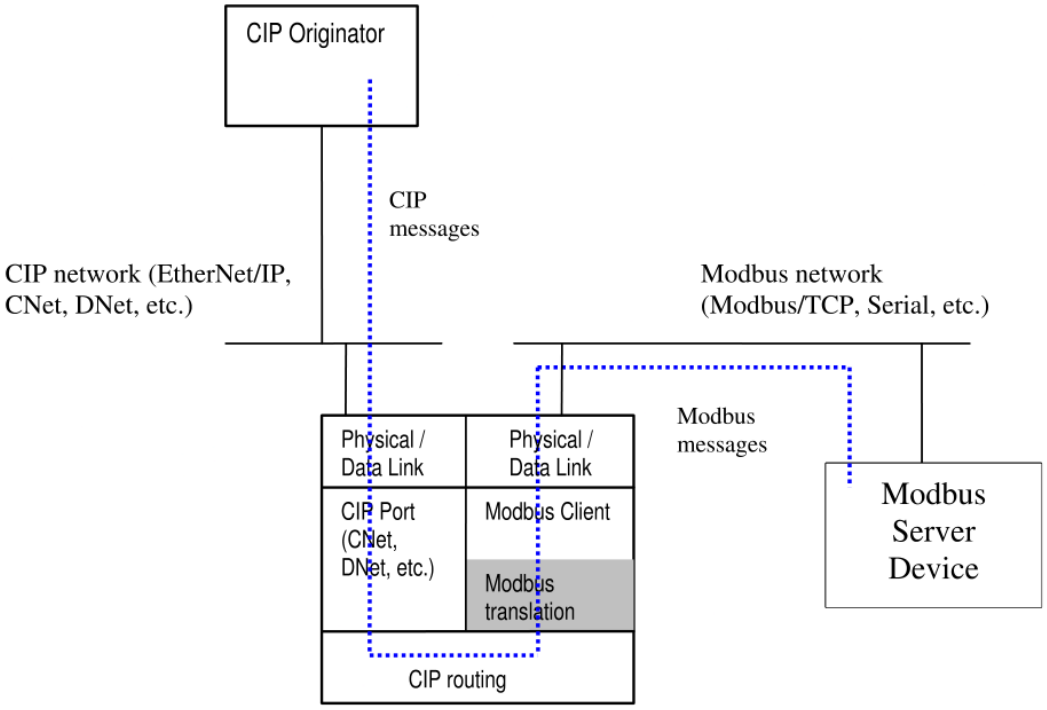
The Modbus Translator can be implemented in a number of different CIP devices, described in the following sections:

- Modbus Translator in a CIP router (e.g., a ControlNet to Modbus/TCP router)
- Modbus Translator in a CIP originator (e.g., an EtherNet/IP scanner)

2-3.2 CIP to Modbus Router

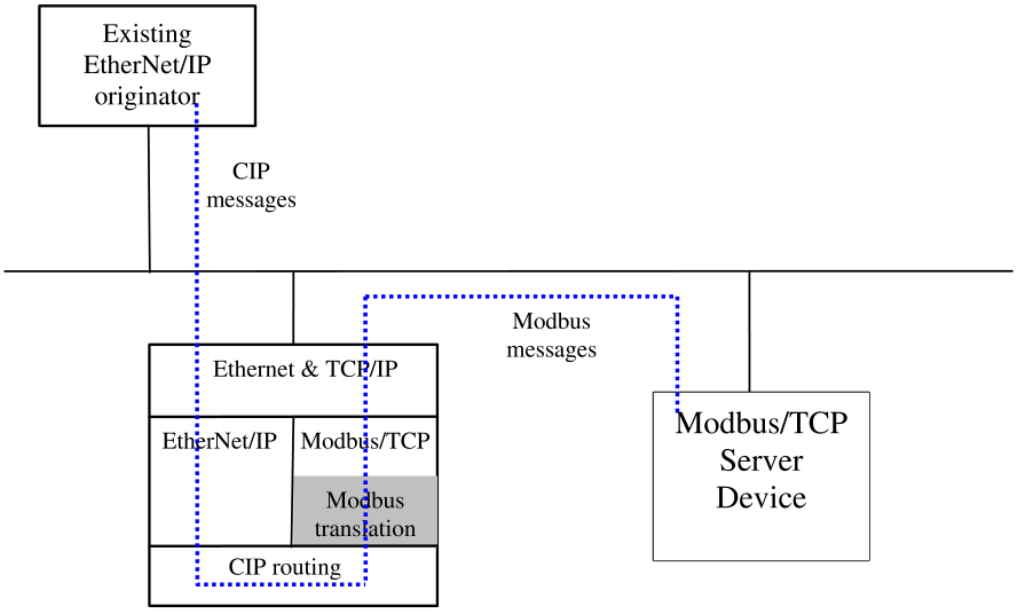
In order to support CIP originators that do not implement the Translator internally, the Translator can be implemented in a standalone CIP to Modbus router. For example a ControlNet-to-Modbus/TCP router would allow ControlNet originators to communicate with Modbus/TCP devices. A DeviceNet-to-Modbus Serial router would allow connectivity between DeviceNet and Modbus Serial devices.

Figure 2-3.1. Translator in a CIP-to-Modbus Router



An EtherNet/IP to Modbus/TCP router would have a single Ethernet interface, and would translate EtherNet/IP communications to Modbus/TCP on the same physical network.

Figure 2-3.2. Translator in an EtherNet/IP-to-Modbus/TCP Router



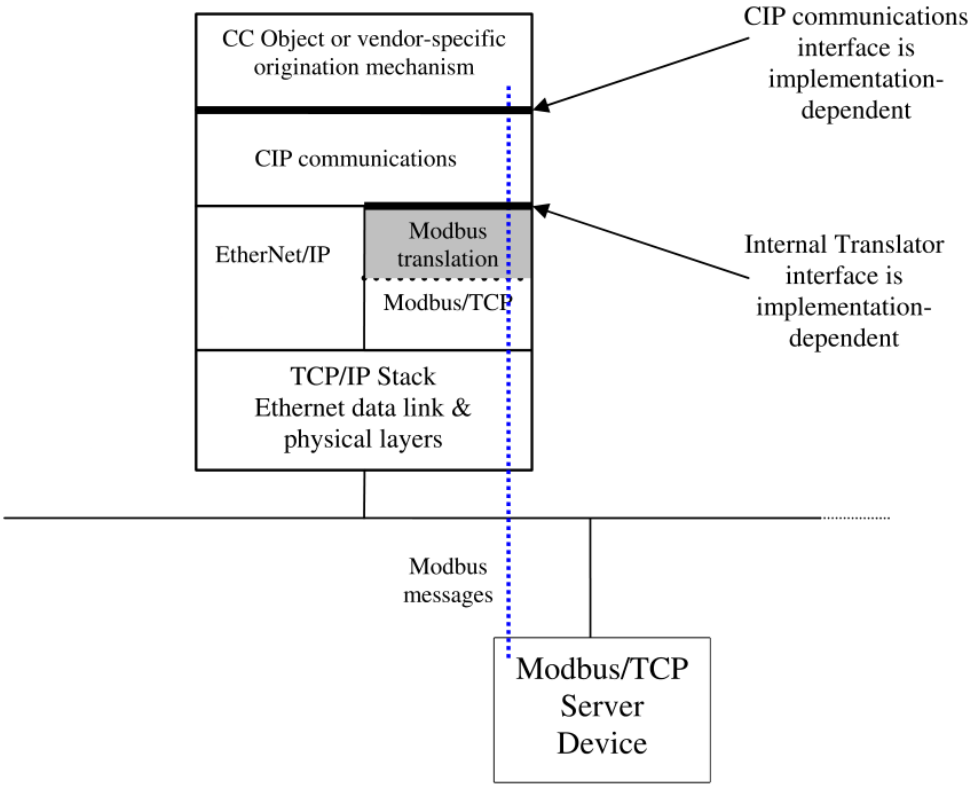
2-3.3 CIP Originator

A CIP originator can implement the Modbus Translator internally. In this case, the CIP application layer accesses the Modbus target device as it would any other CIP device. Internally, the Translator converts between CIP and Modbus.

The interface between the CIP application and the CIP communication layers (including the Translator) is implementation-dependent and not explicitly defined as part of this specification. In general, the CIP communications interface requires constructs to open connections, send and receive connected and unconnected messages, and close connections. When the Modbus Translator is included, the CIP communication constructs are translated to Modbus communications, similar to the way CIP messages are translated to Modbus in a CIP router device (as illustrated in the previous section).

An example of such a device is an EtherNet/IP originator that supports the Connection Configuration Object (CCO). The CCO is an object defined in Volume 1 of the CIP specification, and allows a standard mechanism for configuring connections in CIP originators. Vendors may alternatively use vendor-specific mechanisms for configuring connections in CIP originators.

Figure 2-3.3. Translator in an EtherNet/IP Originator



2-4 Modbus and Modbus/TCP Overview

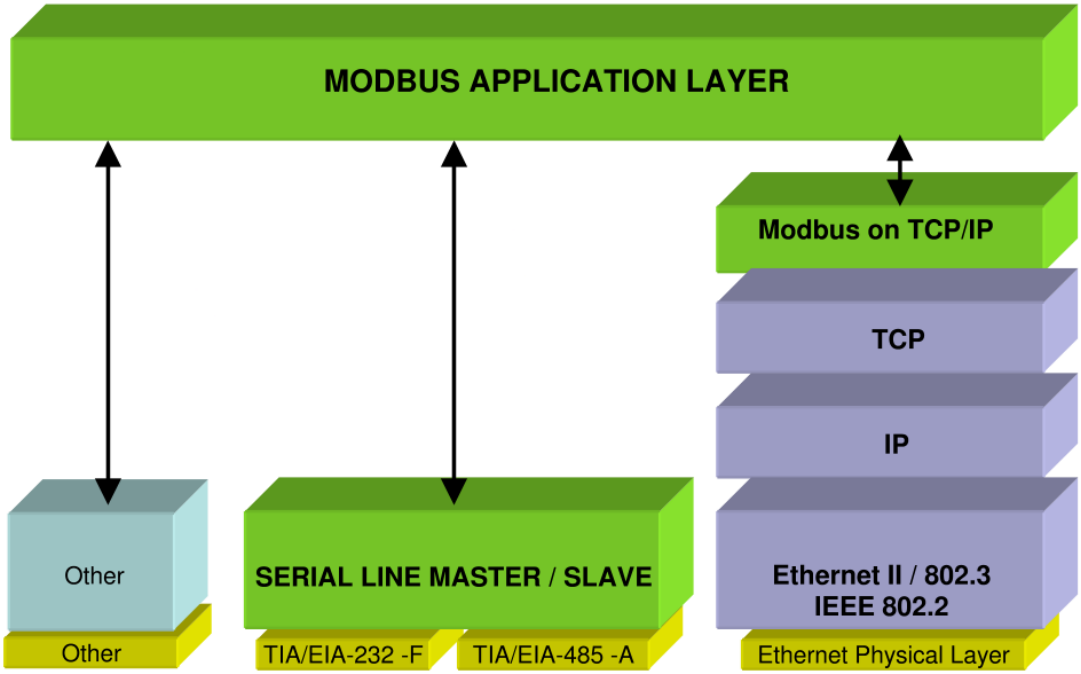
2-4.1 Overview

Modbus is an application-layer protocol used for client/server communications between devices. Modbus can be implemented on a number of different underlying physical networks:

- RS232, RS422, RS485
- TCP/IP over Ethernet
- Modbus Plus, a high speed token passing network
- A variety of other media such as wireless, fiber, radio, cellular, etc.

The Modbus application layer is independent of the physical media over which the messages are sent.

Figure 2-4.1. Modbus Protocol Layers



Refer to the Modbus specifications listed in Chapter 1 for definitive information on the Modbus protocol.

2-4.2 Modbus Data Model

Modbus data consists of four different tables, each with different characteristics, as shown in Table 2-4.1.

Table 2-4.1 Modbus Data Table Characteristics

Table	Data type	Access	Comments
Discrete Inputs	Single bit	Read-only	Data typically provided by an I/O system. This type of data item is commonly used to model binary data manipulated by the server application. Items in this table are intended to be read-only to the client. The read-only integrity of the actual device data is under control of the server application, which can restrict the data exposure to discrete inputs.
Coils	Single bit	Read-write	Data alterable by the client application. These items are useful in the modeling of binary-valued data items in the server device.
Input Registers	16-bit word	Read-only	Data typically provided by an I/O system. This type of item is commonly used to represent analog-valued or generic 16-bit data items manipulated by the server application. Items in this table are intended to be read-only to the client. The read-only integrity of the actual device data is under control of the server application, which can restrict the data exposure to input registers.
Holding Registers	16-bit word	Read-write	Data alterable by a client application. This type of item is commonly used to represent analog-valued or generic 16-bit data items in the device.

There is no required application behavior for each of the tables. Each implementation defines the meaning of the items in each table. Devices may overlay the tables, such that items in different tables refer to the same actual physical data item.

Each table may contain up to 65,536 items. Items may be accessed individually, or as a group starting at a specified item. Note that although items may be accessed as a contiguous group, devices are not required to implement items contiguously.

Each implementation decides how many items in each table to provide (and which tables to provide).

2-4.3 Modbus Protocol

Modbus is a client/server protocol. The Modbus client creates the application data unit (ADU) requesting the server to perform a particular function code. The Modbus ADU contains a function code followed by additional data specific to the function code (e.g., starting register, number of registers, additional sub-function code, etc.).

The Modbus server either performs the specified function code and returns a response indicating success, or returns an error response with a failure code.

The Modbus specification defines the set of public function codes and their associated behavior. Function codes are defined for reading and writing Coils, Discrete Inputs, Holding Registers, etc. Function codes are also defined for obtaining diagnostic data.

The Modbus specifications referenced in Chapter 1 contain the definitive information on the Modbus protocol.

2-5 CIP View of the Modbus Device

2-5.1 Overview

From the perspective of the CIP originator, the Modbus device is viewed as a native CIP device. The Modbus device appears to contain CIP objects and support native CIP communications. The Modbus Translator provides the CIP capability for the Modbus device, transparent to the originating CIP application. The Modbus target device does not actually contain the CIP objects and services.

The following sections give an overview of the CIP capabilities of the Modbus device. Details of the CIP capabilities may be found in the later sections of this specification volume.

2-5.2 CIP Objects

From the CIP perspective, the Modbus server device appears to contain the following CIP objects:

2-5.2.1 Identity Object

The Identity object supports a minimum set of attributes, populated with data from the Modbus device where possible, and with fixed values where necessary.

2-5.2.2 Assembly Object

A defined set of Assembly object instances are translated to the Modbus Holding Register and Input Register tables. By accessing a particular Assembly Object instance, the CIP originator accesses a specific Modbus data item or set of items.

2-5.2.3 Parameter Object

In addition to Assembly object instances, Parameter object instances are translated to Modbus data items. Each Parameter instance corresponds to a single Modbus data item, allowing a CIP originator to read or write a single Modbus Discrete Input, Coil, Holding Register, or Input Register by reading or writing a Parameter instance.

2-5.2.4 Modbus Object

The Modbus object contains attributes representing the Modbus data tables, and services to access items in the data tables. The object includes new CIP services to read and write multiple data items, and a service to allow execution of any Modbus function code. See Chapter 5 for the definition of the Modbus object.

2-5.3 Types of CIP Communications

From the CIP perspective, the Modbus server device appears to support both explicit and implicit CIP communications. Explicit communications can be connected or unconnected, allowing client/server access to the CIP object representation of Modbus data. Implicit communications allows connections to the Assembly object instances, providing implicit access to Modbus Holding Registers and Input Registers (implicit access to Modbus Coils and Discrete Inputs is not supported).

2-5.4 EDS files

Because Modbus devices are not required to change in order to support communications with CIP, an EDS file is not required for each Modbus device. To improve the user experience in accessing Modbus devices from CIP originators, EDS constructs are defined for Modbus devices.

A generic EDS file defines the CIP capability present in any Modbus target device as viewed from CIP, allowing EDS-based CIP originators to communicate with the Modbus device. The generic EDS file specifies the complete Assembly and Parameter instance range, regardless of whether the device supports all of the instances. Using the generic EDS file, a user would typically be required to know and configure which specific Assembly or Parameter instances are supported in the Modbus server device.

A Modbus device may alternatively provide an EDS file specific to the device, allowing better usability with an EDS-capable CIP originator. The specific EDS file would detail the exact Assembly instances, Parameter instances, device identity, and communication parameters supported by the device. With a specific EDS file, a user would have information on which specific instances, corresponding to Modbus data items, the device supports.

2-6 CIP Communications with Modbus Devices

2-6.1 CIP to Modbus Communication Mechanism

The Modbus Translator is central to providing the seamless communication between CIP and Modbus devices. The Translator converts CIP explicit and implicit messages to Modbus requests, and converts Modbus responses to CIP messages.

The following sections describe additional details of the CIP to Modbus communication mechanism.

2-6.2 Connection Path from CIP to Modbus

When a CIP originator sends an unconnected message or establishes a connection, it specifies a connection path to the target device. The connection path indicates the CIP network route to the target device. For communications with Modbus, the connection path indicates that the target device is a Modbus device, informing the Translator that it must translate between CIP and Modbus.

Modbus/TCP and Modbus Serial have specific CIP Port Types as documented in Chapter 3 of Volume 1 (The CIP Common specification). A device that implements Modbus/TCP or Modbus Serial uses a CIP port number that represents the Modbus/TCP or Modbus Serial communication interface.

The port number in the connection path signifies that the connection is to a Modbus target device.

2-6.2.1 Connection Path for Modbus/TCP

The connection path for a Modbus/TCP server device uses the same format as an EtherNet/IP connection path (see Volume 2, Chapter 3).

For example, a connection path from a CCO-based EtherNet/IP originator to a Modbus/TCP server device would be represented as:

2, “192.168.1.10”

Where:

2 is the Modbus/TCP CIP port number in the originating device

“192.168.1.10” is IP address of the target device.

Note that when the connection path is part of the Forward Open or Unconnected Send service, it is encoded according to the rules specified in Volume 1 of the CIP Networks Library. The above notation is commonly used for explanation purposes.

2-6.2.2 Connection Path for Modbus Serial

The connection path for a Modbus serial device is similar to that used for DeviceNet: the CIP port number for the Modbus serial port followed by the 1-byte Modbus server node address. Note that a CIP communication port type is defined for Modbus serial (see Volume 1 of the CIP Networks Library).

For example, a connection path from a CCO-based EtherNet/IP originator to a Modbus serial server device, via an EtherNet/IP-to-Modbus serial router, would be represented as:

2, “192.168.1.10”, 3,100

Where:

2 is the EtherNet/IP CIP port number in the originating device

“192.168.1.10” is the IP address of the EtherNet/IP-to-Modbus serial router

3 is the CIP port number of the Modbus serial port

100 is the node address of the Modbus serial device

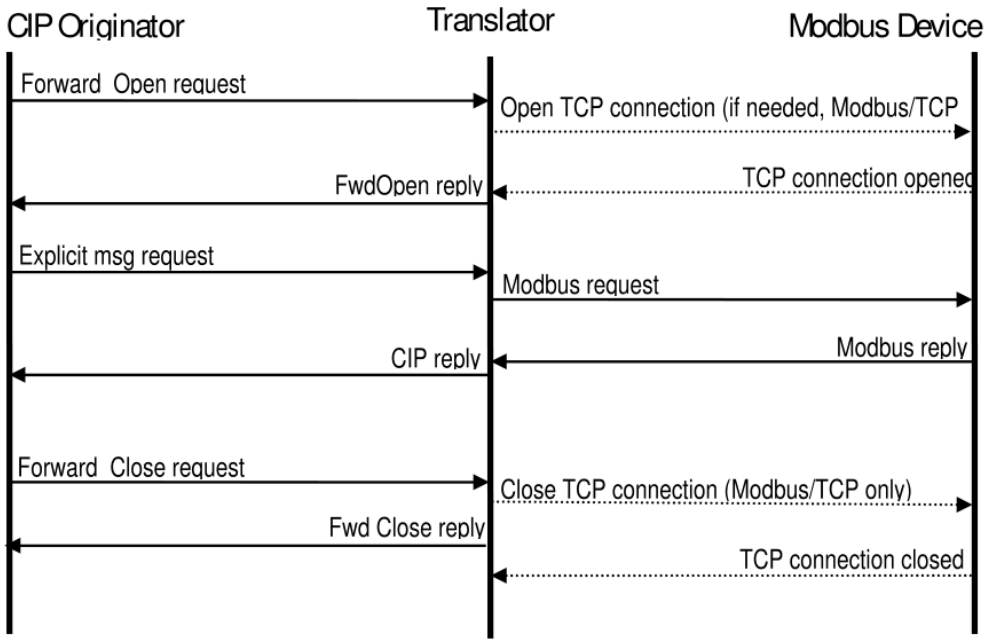
As noted above, when the connection path is part of the Forward Open or Unconnected Send service, it is encoded according to the rules specified in Volume 1 of the CIP Networks Library.

Note: the details of CIP to Modbus Serial translation are not completely specified at present, and will be included in a future version of this specification.

2-6.3 Explicit Communications with Modbus Devices

Figure 2-6.1 illustrates the sequence of explicit communications between a CIP originator and Modbus server device. Note that when the Translator is internal to the originator, the Forward_Open and explicit message requests would be sent via an internal application interface. When the Translator is present in a CIP router, the messages from the CIP originator would appear on the CIP network.

Figure 2-6.1 Explicit Communications Sequence

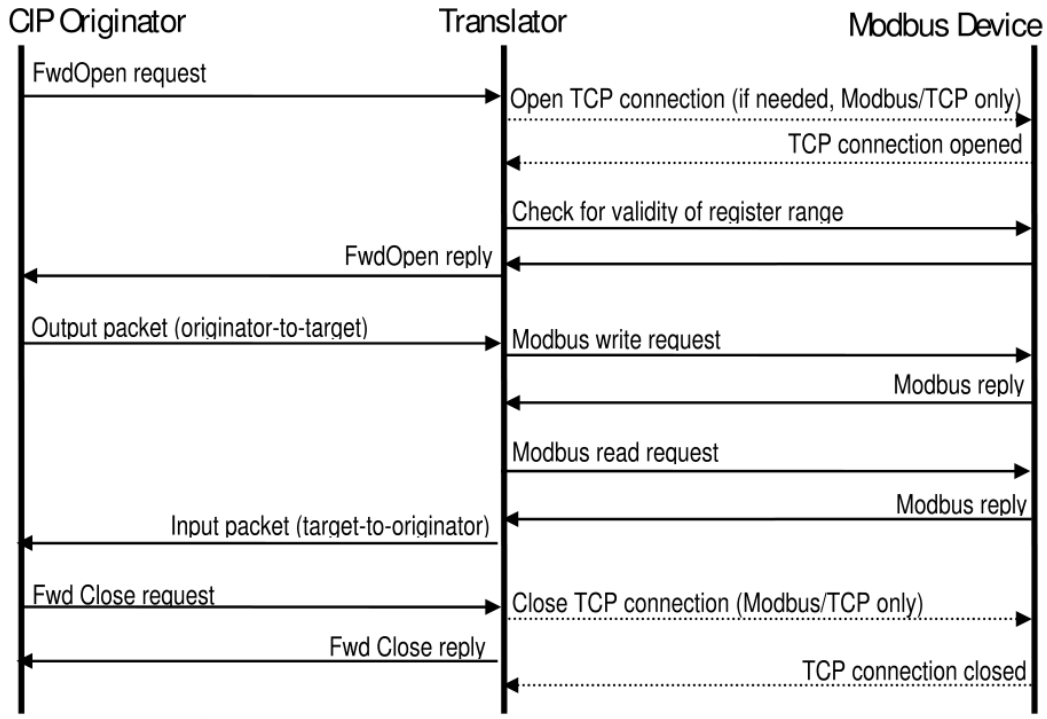


Note: Typically there are many CIP explicit message requests and replies before an originator issues a Forward_Close. It is not recommended for originators to open a connection for a single request then close the connection.

2-6.4 Implicit Communications with Modbus Devices

While Modbus devices do not support implicit communications in the same manner as CIP devices, via the Translator CIP originators can establish implicit connections to access register data in the Modbus server device. The Translator converts the implicit messages from the originator (outputs) into Modbus write requests, issues Modbus read requests and converts the responses to implicit messages to the originator (inputs).

Figure 2-6.2. Implicit Communications Sequence



Note: The implicit CIP connection typically persists over many cycles of Input and Output packets. A Forward_Close request is not typically sent after sending one Output packet and receiving one Input packet.

2-7 Summary of the CIP Functions Supported by the Translator

Below is a brief list of CIP Functions which are supported by the Translator as described throughout this document.

- Unconnected Explicit Messaging
- Connected Class 3 Explicit Messaging
- Connected Class 1 Implicit Messaging with cyclic triggering
- Connected Implicit works with transport class 3
- 32-bit Run/Idle headers in the O->T direction, modeless in the T->O direction
- Unicast in the T->O direction
- Unicast in the O->T direction
- Multicast (with a matching RPI) in the T->O direction
- Data Segment (for the Modbus devices with their Configuration Data in a contiguous item block)
- Implicit ExclusiveOwner and InputOnly (there is no notion of redundant ownership)
- Parameter Objects
- Assembly Objects
- Identity Objects
- Modbus Object

This page intentionally left blank

Volume 7: Integration of Modbus Devices into the CIP Architecture

Chapter 3: Integrating Modbus Devices with CIP

Contents

3-1 Intended Target Audience

This chapter is intended to be used by persons with existing Modbus devices and designers of new Modbus devices interested in integrating those devices into CIP. Users of existing Modbus device are advised to read this chapter to determine how best to apply the capabilities of the Modbus device for use with CIP. When designing a new Modbus device the designer is advised to use this chapter to determine the features best suited for the new device in order to operate more seamlessly with CIP.

3-2 Modbus Device Recommendations for CIP Integration

There are no required Modbus function codes for a Modbus device to operate in a CIP environment. Since every Modbus device is different and supports a different set of Modbus function codes it is difficult to require specific Modbus function codes. For example a simple sensor will only support a read function code and a simple actuator will only support a write function code. Other more complex devices may support more functions but there is no guarantee a Modbus device will support all of the recommended function codes. However recommendations are made in this section to allow a Modbus device to more easily integrate with CIP. Vendors designing new Modbus products or updating existing firmware are strongly encouraged to support at least this recommended function list.

The following Modbus function codes are recommended to be supported in a Modbus device for operating with CIP:

- Modbus FC 03 (0x03) Read Holding Registers

Allows reading of Holding Registers via I/O connections or via explicit messages to the Assembly and Parameter Objects.

- Modbus FC 16 (0x10) Write Multiple Registers

Allows writing of Holding Registers via I/O connections or via explicit messages to the Assembly and Parameter Objects. When there is a need to write holding registers, writing to a Modbus device will be more efficient than single register writes, reducing the number of required writes.

- Modbus FC 23 (0x17) Read/Write Multiple Registers

To reduce the number of calls and resources utilization, while improving data coherency, FC 23 can be used to exchange the CIP IO connection information. This can be an addition or an alternative to FC 03 and FC 16, when both read and write are needed.

- Modbus FC 43/14 (0x2B/0x0E) Read Device Identification

Using this function code will allow the originator to identify the device through CIP. This function allows the reading of a device in order to retrieve the identification and information relative to the physical and functional description of the device.

3-3 Configuration for Modbus Devices

3-3.1 TCP/IP Configuration

CIP does not specify the method that Modbus devices shall use for configuration. However a Modbus/TCP device could use the following means to provide the configuration for TCP/IP functions such as an IP addresses, etc. One of the following methods could be used for this purpose:

- BOOTP
- DHCP
- Locally through an integrated HMI
- Use of the Modbus Write Registers (through an attached serial port)

Since DHCP is recommended for EtherNet/IP devices, DHCP client support is recommended where possible for Modbus/TCP devices.

3-3.2 Modbus Device Configuration

If a Modbus device supports contiguous register for configuration then they can receive configuration upon the opening of a CIP I/O connection. Vendors are encouraged not to mix read-only and read-write registers or to create “holes” of unsupported registers within configuration areas.

3-4 Modbus Device Identification

As noted above, it is recommended that Modbus devices support Modbus Function Code 43/14. The Modbus device should return a “ProductCode string” that is consistent with the “ProductName attribute” definition of the CIP Identity Object. This allows the ProductCode string to be mapped to the ProductName attribute as described in Chapter 5.

3-5 Modbus Device Performance

It is recommended that a Modbus device locate any real-time registers in a contiguous area. This will allow the use of the recommended Modbus function codes listed in section 3-2 in a more efficient manner. When registers are contiguous, a single Modbus request can be used to read or write the group of registers. If needed a redirection table could be used as a mechanism to allow such grouping of the registers into a contiguous area.

3-6 Modbus Device Behavior

It is recommended that the Modbus device reverts to a fallback state when the network communication is not present for a measured amount of time. This measured time will vary with the type of device and must be reasonable for that device.

The fallback state depends on the type of Modbus device and the application of the device. There are three possible states into which a device can fallback.. The device can hold its current values, the device can turn off the outputs or the device can turn on the outputs.

The same recommendation applies to Modbus devices when the device is on the network and is powered up but has not yet made a connection to the controlling entity.

3-7 EDS Files

EDS files can be created for Modbus devices to help CIP-based configuration tools inform the user of the capabilities of the Modbus device. For Modbus devices, the EDS file can contain the specific information about the CIP representation of the device's Modbus data. For example, an EDS file can specify which registers are provided by specifying which Parameter and Assembly instances are provided. Writing a custom EDS file for a device can help users more easily configure their communications access to Modbus devices from CIP. Detailed information about EDS files is covered in Chapter 7 of this document. Identity Objects and parameter information will be included in Chapter 5.

3-8 User Considerations

This section highlights items the user should consider when using Modbus with CIP. It is not meant to tell the user how to do the design but provide items that could be possible issues when using Modbus devices inside of CIP.

The user needs to realize that CIP is not programmatically aware of the Modbus timeout. The Modbus timeout accommodation can only be explicit and specific to a device.

Modbus devices may not have contiguous register space. For a scenario description see Chapter 7, Section 7-2.2.3.

Exclusive ownership cannot be enforced with Modbus devices. Activities that rely on exclusive ownership may not perform as expected.

Modbus targets may not support all of the function codes required for I/O and this may result in I/O connections not being supported.

Data for I/O or explicit messaging to the Assembly Object or Parameter needs to be exchanged with the Modbus Translator in CIP byte order. The Modbus Translator will handle byte swapping to the Modbus device. See Chapter 6, Section 6-2.3.

The user should be aware that reads and writes to the same register in a Modbus device may address different memory locations on the device. For a scenario description see Chapter 7, Section 7-2.2.2.

For a Modbus device, the capability of interacting with a block of data does not imply the possibility to individually manipulate the data within that block.

When doing an I/O connection to a Modbus device, the user needs to be aware that the connection size indicates how many registers will be read or written on the device.

It is important for the user to get a detailed data map of the Modbus tables implemented on the Modbus device from the vendor.

The CIP refresh rate should be configured in a way that is consistent with the target Modbus device. Care is needed to be sure CIP refresh rate is not faster than the rate at the target Modbus device can consume the corresponding writes.

If the CIP refresh rate is not consistent with the target Modbus device then the user needs to be aware of possible multiple writes from the CIP side through the Modbus translator to the target Modbus device. Since there is no correlation between the CIP sequence number and the Modbus transaction identifier the target Modbus device could receive multiple writes where each write will have a new Modbus transaction identifier.

Modbus devices require independent read function in order to work with CIP I/O connections. Devices that otherwise would use Function Code 23 only are recommended to also implement the read Function Code 03,

In both CIP and Modbus, a user expects to be able to send a message from the originator to the target within a given RPI time. In CIP, if the target does not produce on its expected RPI, the originator will continue to send messages on the RPI. Regardless of the RPI, it will send the number of messages equal to the timeout multiplier of the connection, typically four, before timing out.

Due to TCP characteristics, Modbus/TCP does not support this behavior. Instead, it uses the standard TCP method to insure a message is sent to the target device. If a TCP packet is lost (no reception of TCP acknowledgment), TCP first retransmission occurs around one second after the first transmission. During this time, additional updates of the production sent by the CIP Originator will not be sent by TCP to the Modbus/TCP device. The CIP connection will time out on the CIP Originator side in most cases.

Volume 7: Integration of Modbus Devices into the CIP Architecture

Chapter 4: Object Model

Contents

4-1 Introduction

This chapter contains additions to the CIP object model that are specific to Modbus device integration. At this time, no such additions exist.

This page is intentionally left blank

Volume 7: Integration of Modbus Devices into the CIP Architecture

Chapter 5: Object Library

Contents

5-1 Introduction

This chapter defines a new object and additions/restrictions to existing objects for use when interfacing to Modbus devices. Since Modbus devices do not contain CIP objects, a translation function shall translate CIP requests targeted to Modbus devices. Part of this translation involves exposing CIP objects as if they reside in the Modbus target device.

5-2 Identity Object

Class Code: 01hex

5-2.1 Instance Attributes

Table 5-2.1 Modbus Specific Identity Instance Attributes

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	Get	Vendor ID	See Volume 1, Chapter 5		Translation function shall return 65534
2	Required	Get	Device Type	See Volume 1, Chapter 5		Translation function shall return 0x28
3	Required	Get	Product Code	See Volume 1, Chapter 5		Translation function shall return 0
4	Required	Get	Revision	See Volume 1, Chapter 5		Translation function shall return 0
5	Required	Get	Status	See Volume 1, Chapter 5		Translation function shall return 0
6	Required	Get	Serial Number	See Volume 1, Chapter 5		Translation function shall return 0
7	Required	Get	Product Name	See Volume 1, Chapter 5		Translation function shall return the first 32 characters of the Modbus Product Code[2, 3]
8 – 17	Not Allowed	See Volume 1, Chapter 5				
18	Conditional[1]	Get	Modbus Identity Info	STRUCT of:	Device Identification	See "Semantics" section
				SHORT_ STRING	VendorName[2]	
				SHORT_ STRING	ProductCode[2]	
				SHORT_ STRING	MajorMinorRevision[2]	
				SHORT_ STRING	VendorUrl[2]	
				SHORT_ STRING	ProductName[2]	
				SHORT_ STRING	ModelName[2]	
				SHORT_ STRING	UserAppName[2]	
All Other Attributes	Not Allowed	See Volume 1, Chapter 5				

1 This attribute is required if the Modbus device supports Function Code 0x2B MEI 0x0E.
2 See Modbus Function Code 0x2B, MEI 0x0E.
3 If the Modbus target device does not support Function Code 0x2B, MEI 0x0E, then this attribute shall contain the string "Unknown Modbus Device".

5-2.1.1 Modbus Device Identity Info

The translation function shall attempt to populate this attribute based on information obtained using the Modbus Read Device Identification function (Function Code 0x2B, MEI 0x0E). Any device identification objects not supported by the target Modbus device shall be represented by a zero length string in the related field within the attribute.

The translation function shall attempt to get all regular device identification using Read Device ID Code 0x02. If the device supports both this function and Device ID Code, then attribute 7 and all fields within attribute 18 will be populated with the values returned. If the device only supports the Basic Identification, then attribute 7 and the first three fields of attribute 18 will be populated with the values returned and the remaining fields in attribute 18 shall be set to zero length strings.

If the device does not support the Read Device Identification function, no device identification is available for the specified target Modbus device and the translation function shall:

- Return the string value of “Unknown Modbus Device” for attribute 7.
- Return an error with a general status of Attribute Not Supported (0x0E) and no extended status for attribute 18.

5-3 Assembly Object

Class Code: 04hex

The Assembly Object provides a translation into the Modbus data item tables in the Modbus target device. The Assembly instance ranges are divided up among the four primary Modbus data tables as shown in the table below. Each range consists of 65,536 instances (one for each data item in each Modbus data table).

Table 5-3.1 Assembly Instance ID Ranges

Range	Meaning	Quantity
0x00000001 – 0x00010000	Holding Registers (4X)	65,536
0x00010001 – 0x00020000	Input Registers (3X)	65,536
0x00020001 – 0x00030000	Coils (0X)	65,536
0x00030001 – 0x00040000	Discrete Input (1X)	65,536

All translated assembly instances are Static Assemblies. Dynamic Assemblies are not supported in Modbus target devices.

The first member in the member list of each Assembly instance is the Modbus data item in the respective data table as indicated by the instance number. Each successive member in the list is the next data item in the data table. The member list continues until the assembly size reaches:

- 250 bytes for a Get (read) request or 246 bytes for a Set (write) request
- To the end of the table if less than 250 bytes (for a Get request) or 246 bytes (for a Set request) remain in the table.

See Volume 1, Chapter 6, Device Profiles for more information.

5-3.1 Class Attributes

The following table indicates modifications to the Need in Implementation for class attributes of the Assembly Object.

Table 5-3.2 Modifications to Assembly Class Attributes

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Optional	Get	Revision	See Volume 1, Chapter 5		
2 thru 7	Not Allowed	See Volume 1, Chapter 5				

5-3.2 Instance Attributes

The following table indicates modifications to the Need in Implementation for instance attributes of the Assembly Object.

Table 5-3.3 Modifications to Assembly Instance Attributes

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 - 2	Not Allowed	See Volume 1, Chapter 5				

5-3.3 Common Services

The following table indicates modifications to the common services of the Assembly Object.

Table 5-3.4 Modifications to Assembly Common Services

Service Code	Need in Implementation				Service Name	Description of Service
	Static Assembly		Dynamic Assembly			
	Class	Instance	Class	Instance		
$0E_{hex}$	Conditional[1]	n/a	n/a	n/a	Get_Attribute_Single	See Volume 1, Chapter 5
08_{hex}	n/a	n/a	n/a	n/a	Create	
10_{hex}	n/a	n/a	n/a	n/a	Set_Attribute_Single	
09_{hex}	n/a	n/a	n/a	n/a	Delete	
$1A_{hex}$	n/a	n/a	n/a	n/a	Insert_Member	
$1B_{hex}$	n/a	n/a	n/a	n/a	Remove_Member	
18_{hex}	n/a	Required	n/a	n/a	Get_Member[2]	
19_{hex}	n/a	Required	n/a	n/a	Set_Member[2]	

1 Required if Class Attribute 1 is supported.
2 Translator shall return error code 0x28 (Invalid Member) if the Modbus device returns exception code 0x02.

5-4 Connection Manager Object

Class Code: 06hex

The following sections identify differences in the requirements for the Connection Manager object within Modbus target devices. There are no modifications for the class level attributes and services.

5-4.1 Instance Attributes

The following table indicates modifications to the Need in Implementation for instance attributes of the Connection Manager Object.

Table 5-4.1 Modifications to Connection Manager Class Attributes

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 13	Not Allowed	See Volume 1, Chapter 3				

5-4.2 Instance Specific Services

The following table indicates modifications to the instance specific services of the Connection Manager Object.

Table 5-4.2 Modifications to Connection Manager Instance Attributes

Service Code	Need in Implementation	Service Name	Description of Service
$4E_{hex}$	Required	Forward_Close	See Volume 1, Chapter 3
52_{hex}	Conditional[1]	Unconnected_Send	
54_{hex}	Required	Forward_Open	
56_{hex}	Not Allowed	Get_Connection_Data	
57_{hex}	Not Allowed	Search_Connection_Data	
$5A_{hex}$	Not Allowed	Get_Connection_Owner	
$5B_{hex}$	Not Allowed	Large_Forward_Open	

1 Required if the device is a Modbus/TCP Gateway.

5-5 Parameter Object

Class Code: 0Fhex

The Parameter Object provides access to an individual data item within the Modbus data tables in the Modbus target device. The Parameter instance ranges are divided up among the four primary Modbus data tables as shown in the table below. Each range consists of 65,536 instances (one for each data item in each Modbus data table).

Table 5-5.1 Parameter Instance ID Ranges

Range	Meaning	Quantity
0x00000001 – 0x00010000	Holding Registers (4X)	65,536
0x00010001 – 0x00020000	Input Registers (3X)	65,536
0x00020001 – 0x00030000	Coils (0X)	65,536
0x00030001 – 0x00040000	Discrete Input (1X)	65,536

5-5.1 Class Attributes

The following table indicates modifications to the Need in Implementation for class attributes of the Parameter Object.

Table 5-5.2 Modifications to Parameter Class Attributes

Attribute ID	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
2 thru 7	Not Allowed	See Volume 1, Chapter 5				

5-5.2 Instance Attributes

The Parameter instance attributes are modified for Modbus target devices as described in the table below.

Table 5-5.3 Modifications to Parameter Instance Attributes

Number	Need in Implementation	Stub/ Full	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1	Required	N/A	Set[1]	Parameter Value	word or BOOL[2]	Value of parameter. If the parameter instance indicates a data item in a read only Modbus table, then this attribute is Get only.	
All Other Attributes	Not Allowed	See Volume 1, Chapter 5					

1 If the parameter instance indicates a data item in a read only Modbus table, then this attribute is Get only.

2 If the parameter instance indicates a data item in a Modbus word table, the data is returned as a 16 bit entity; the actual data type of the value is unknown. If the parameter instance indicates a data item in a Modbus bit table, the data type is BOOL.

5-5.3 Common Services

The following table indicates modifications to the common services of the Parameter Object.

Table 5-5.4 Modifications to Parameter Common Services

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
$0D_{hex}$	n/a	n/a	Apply_Attributes	See Volume 1, Chapter 5
$0E_{hex}$	Conditional[1]	Required	Get_Attribute_Single	
10_{hex}	n/a	Conditional[2]	Set_Attribute_Single	
01_{hex}	n/a	n/a	Get_Attributes_All	
05_{hex}	n/a	n/a	Reset	
15_{hex}	n/a	n/a	Restore	
16_{hex}	n/a	n/a	Save	
18_{hex}	n/a	n/a	Get_Member	

1 Required if Class Attribute 1 is supported.
2 Required if the parameter represents a Holding Register or Coil

5-6 Modbus Object

Class Code: 44hex

The Modbus object provides an interface to the data and functions within a target Modbus device.

5-6.1 Class Attributes

Table 5-6.1 Modbus Object Class Attributes

Number	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
1 thru 7	These class attributes are optional and are described in Volume 1, Chapter 4.					

5-6.2 Instance Attributes

Table 5-6.2 Modbus Object Instance Attributes

Number	Need in Implementation	Access Rule	Name	Data Type	Description of Attribute	Semantics of Values
No Instance Attributes are Defined						

5-6.3 Common Services

The Modbus Object provides the following Common Services:

Table 5-6.3 Modbus Object Common Services

Service Code	Need in Implementation		Service Name	Description of Service
	Class	Instance		
$0E_{hex}$	Conditional[1]	N/A	Get_Attribute_Single	Returns the contents of the specified attribute.

1 This service is required if any of the class attributes are supported.

5-6.4 Object–specific Services

The Modbus Object provides the following Object–specific services:

Table 5-6.4 Modbus Object Object-specific Services

Service Code	Need in Implem		Service Name	Description of Service
	Class	Instance		
$4B_{hex}$	N/A	Required	Read Discrete Inputs	Reads one or more contiguous discrete input(s).
$4C_{hex}$	N/A	Required	Read Coils	Reads one or more contiguous coil(s).
$4D_{hex}$	N/A	Required	Read Input Registers	Reads one or more contiguous input register(s).
$4E_{hex}$	N/A	Required	Read Holding Registers	Reads one or more contiguous holding register(s).
$4F_{hex}$	N/A	Required	Write Coils	Writes one or more contiguous coil(s).
50_{hex}	N/A	Required	Write Holding Registers	Writes one or more contiguous holding register(s).
51_{hex}	N/A	Required	Modbus Passthrough	Provides encapsulation of any public or private Modbus function.

5-6.4.1 Read Discrete Inputs Service

This service reads one or more discrete inputs from the Modbus Discrete Inputs table. This service results in the translation function issuing a Read Discrete Inputs function (function code 0x02) on Modbus.

Table 5-6.5 Read Discrete Inputs Request Service Parameters

Name	Data Type	Description of Request Parameter	Semantics of Values
Starting Address	UINT	Offset in table to begin reading from.	
Quantity of Inputs	UINT	Number of inputs to read	

Table 5-6.6 Read Discrete Inputs Response Service Parameters

Name	Data Type	Description of Request Parameter	Semantics of Values
Input Values	ARRAY of octet	Input values read. Each input is packed as a bit within a byte.	

5-6.4.2 Read Coils Service

This service reads one or more coils from the Modbus Coils table. This service results in the translation function issuing a Read Coils function (function code 0x01) on Modbus.

Table 5-6.7 Read Coils Request Service Parameters

Name	Data Type	Description of Request Parameter	Semantics of Values
Starting Address	UINT	Offset in table to begin reading from.	
Quantity of Coils	UINT	Number of coils to read	

Table 5-6.8 Read Coils Response Service Parameters

Name	Data Type	Description of Request Parameter	Semantics of Values
Coil Status	ARRAY of octet	Coil values read. Each coil is packed as a bit within a byte.	

5-6.4.3 Read Input Registers Service

This service reads one or more input registers from the Modbus Input Register table. This service results in the translation function issuing a Read Input Registers function (function code 0x04) on Modbus.

Table 5-6.9 Read Input Registers Request Service Parameters

Name	Data Type	Description of Request Parameter	Semantics of Values
Starting Address	UINT	Offset in table to begin reading from.	
Quantity of Input Registers	UINT	Number of input registers to read	

Table 5-6.10 Read Input Registers Response Service Parameters

Name	Data Type	Description of Request Parameter	Semantics of Values
Input Register Values	ARRAY of 16-bit word[1]	Input register values read.	

1 The data is returned as 16 bit entities for each register. The actual data type of the values is unknown.

5-6.4.4 Read Holding Registers Service

This service reads one or more holding registers from the Modbus Holding Register table. This service results in the translation function issuing a Read Holding Registers function (function code 0x03) on Modbus.

Table 5-6.11 Read Holding Registers Request Service Parameters

Name	Data Type	Description of Request Parameter	Semantics of Values
Starting Address	UINT	Offset in table to begin reading from.	
Quantity of Holding Registers	UINT	Number of holding registers to read	

Table 5-6.12 Read Holding Registers Response Service Parameters

Name	Data Type	Description of Request Parameter	Semantics of Values
Holding Register Values	ARRAY of 16-bit word[1]	Holding register values read.	

1 The data is returned as 16 bit entities for each register. The actual data type of the values is unknown.

5-6.4.5 Write Coils Service

This service writes one or more coils to the Modbus Coils table. This service results in the translation function issuing a Write Multiple Coils function (function code 0x0F) on Modbus. If the function fails with a Modbus Exception code of 0x01 (Illegal Function) and the Quantity of Outputs field is 1, the translation function shall then attempt to use the Write Single Coil function (function code 0x05).

Table 5-6.13 Write Coils Request Service Parameters

Name	Data Type	Description of Request Parameter	Semantics of Values
Starting Address	UINT	Offset in table to begin writing to.	
Quantity of Outputs	UINT	Number of output coils to write.	
Output Values	ARRAY of octet	Output coil values	

Table 5-6.14 Write Coils Response Service Parameters

Name	Data Type	Description of Request Parameter	Semantics of Values
Starting Address	UINT	Offset in table where writing began.	
Quantity of Outputs	UINT	Number of outputs forced.	

5-6.4.6 Write Holding Registers Service

This service writes one or more holding registers to the Modbus Holding Registers table. This service results in the translation function issuing a Write Multiple Registers function (function code 0x10) on Modbus. If the Write Multiple Registers function fails with a Modbus Exception code of 0x01 (Illegal Function) and the Quantity of Outputs field is 1, the translation function shall then attempt to use the Write Single Register function (function code 0x06).

Table 5-6.15 Write Holding Registers Request Service Parameters

Name	Data Type	Description of Request Parameter	Semantics of Values
Starting Address	UINT	Offset in table to begin writing to.	
Quantity of Outputs	UINT	Number of holding registers to write.	
Output Values	ARRAY of 16-bit word [1]	Holding register values	

1 The data is returned as 16 bit entities for each register. The actual data type of the values is unknown.

Table 5-6.16 Write Holding Registers Response Service Parameters

Name	Data Type	Description of Request Parameter	Semantics of Values
Starting Address	UINT	Offset in table where writing began.	
Quantity of Outputs	UINT	Number of outputs forced.	

5-6.4.7 Modbus Passthrough Service

The Modbus Passthrough service provides a way for a client to initiate a specific Modbus function to a target Modbus device. The Modbus request and response are encapsulated in the CIP request and response service data fields with no modification. The translation function shall not attempt to perform any endian conversion on the bytes in the data stream (either request or response).

Table 5-6.17 Modbus Passthrough Request Service Parameters

Name	Data Type	Description of Request Parameter	Semantics of Values
Function Code	USINT	Function code of the Modbus request.	1 – 127
Data Request	Array of octet	Parameter data for the Modbus function request. This may include sub-function codes.	Maximum of 252 bytes.

Table 5-6.18 Modbus Passthrough Response Service Parameters

Name	Data Type	Description of Request Parameter	Semantics of Values
Function Code or Exception Code	USINT	Function code of the Modbus response.	1 – 255
Data Response	ARRAY of octet	Parameter data for the Modbus function response. This may include sub-function codes.	Maximum of 252 bytes.

5-6.5 Error Checking

Errors on request parameters that can be detected by the translation function shall result in a CIP error response generated by the translation function. These error conditions do not generate any Modbus traffic.

The following conditions will result in a General Error Code of 0x20 (Invalid Parameter):

- Requesting zero (0) quantity of data values
- Requesting more data values than can be supported by Modbus; for example requesting to read 500 holding registers.
- Any request where combining the start of the data offset with the quantity of data would cause an overflow, for example requesting to read 100 registers starting at offset hex 0xFFFF.

The following condition will result in a General Error Code of either 0x13 (Not Enough Data) or 0x15 (Too Much Data):

- Write requests where the bytes of attached data do not properly match the quantity of data values being written; for example, requesting to write 4 registers but supplying any number of data bytes other than 8

All CIP requests which create valid Modbus requests shall be sent to the targeted Modbus device. Refer to Chapter 10, Explicit Messaging Error Mapping, for the exact Modbus to CIP translation to use if the target device returns an exception code.

This page is intentionally left blank

Volume 7: Integration of Modbus Devices into the CIP Architecture

Chapter 6: Device Profiles

Contents

6-1 Modbus Device

Device Type: 28 hex

This chapter describes information to be used by developers of devices that support the translation function. CIP Originators communicate to Modbus devices using an abstraction of a Modbus device that is represented as a CIP Object Model.

The objects in this model provide interfaces by which the Originator can communicate to the target Modbus device for data access, device configuration, and device management activities through a device that supports the CIP-Modbus translation function. This translation function may be embedded within a CIP Originator, with a module in the CIP Originator's chassis or in a separate device on the network.

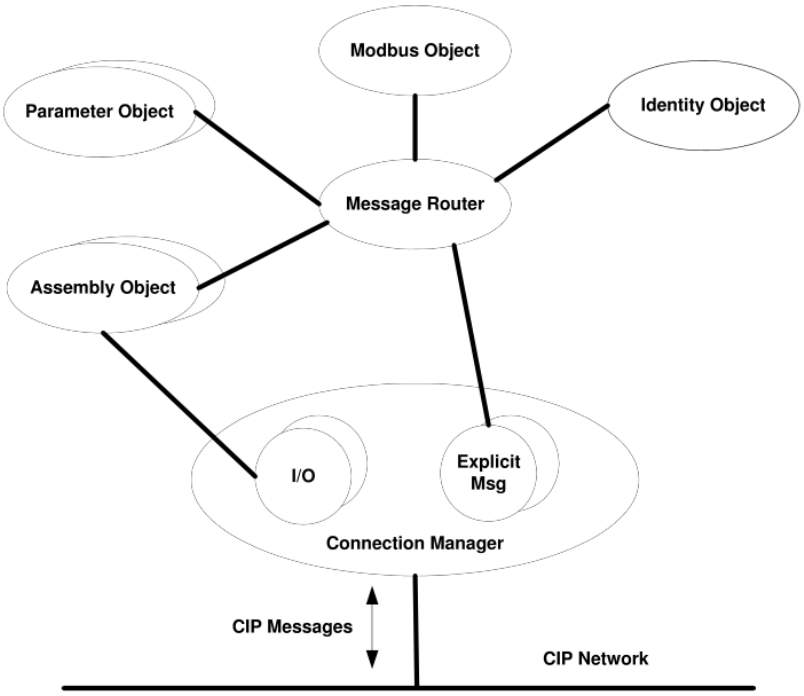
6-2 Object Model

The CIP object model that represents a Modbus device is shown in Figure 6-2.1 below. This model is based on the Generic Device Profile found in Volume 1, Chapter 6-8 and is modified by adding the Modbus Object and removing any network specific object(s).

The Modbus Device Profile in this chapter makes references to objects and functions that may be identified as optional, and/or contain optional capabilities within the objects themselves (e.g.: a required object with an attribute defined as optional in the object definition in Chapter 5). This means that the functionality being described may not be supported in al Modbus devices. Whether the objects/functions that can be represented in CIP are supported by the defined translated function in the Modbus device will generally be indicated within the Modbus device's specific EDS, if provided.

The CIP-Modbus Translator provides the object interfaces which handle inbound CIP messages from the CIP network. Therefore, implementations of the translation function shall support all optional capabilities of this object model, unless otherwise explicitly stated, so that it is able to provide the proper translation when the associated capability does exist in the Modbus device.

Figure 6-2.1 Modbus Device Profile



The Assembly, Parameter, and Modbus objects provide access to the Modbus Tables of the target device. These objects are accessed via the CIP Network in the usual way by means of:

- CIP Unconnected Messaging,
- Implicit Connections, and
- Explicit Connections.

The Modbus Object provides Modbus Table specific services. See Volume 7 Chapter 5 for additional detail regarding the Modbus Object and the Assembly and Parameter Objects when used for Modbus Devices.

All CIP data access mechanisms have as their target objects the Modbus, Parameter and Assembly objects that are used to model the data provided by the Modbus target device. These objects map to or provide access to the Modbus Tables of the device.

Implicit Connections are defined only to Assembly Objects whose Instance IDs are in the range defined for holding registers or input registers. Assembly Object, Parameter Objects and the Modbus Object can be accessed using explicit messages, both connected and unconnected.

A description of CIP Messaging is found in Volume 1 Chapter 2 “Messaging Protocol”.

6-2.1 Minimum Objects Required

The Object Model in Figure 6-2.1 represents the CIP objects of any Modbus Device seen from the CIP Network. Refer to the object library chapters, i.e.: Volume 1, Chapter 5 and Volume 7, Chapter 5 for more details about these objects. The table below indicates the number of instances of each objects and whether the class is required or not.

A description of the definition of the Parameter and Assembly objects for specific Modbus device conventions and their associated connections will be found in Chapter 7.

Table 6-2.1 Minimum Objects Required

Object Class	Optional / Required	# of Instances	Note
Identity	Required	1	
Message Router	Required	1	
Connection Manager	Required	1	
Modbus	Required	1	
Parameter	Required	1 .. n	Device Specific
Assembly	Required	1 .. n	Device Specific

Instance IDs that relate to Parameter and Assembly objects mapped to the Modbus Tables must follow the range conventions found in Chapter 5-3.

6-2.2 Objects That Affect Behavior

The objects for this device affect the device's behavior as shown in the table below.

Table 6-2.2 Object Effect on Behavior

Object	Effect on behavior
CIP Common Required	See Volume 1, Chapter 6-2.2 for details.
Assembly	Defines Modbus tables mapping
Parameter	Provide a public interface to make explicit access to a single Modbus data Item.
Modbus	Provides a public interface to make any Modbus request to the Modbus device and provides read and write access to multiple Modbus data items.

6-2.3 Object Interfaces

The objects in this device have the interfaces listed in the following table:

Table 6-2.3 Object Interfaces

Object	Interfaces
CIP Common Required	See Volume 1, Chapter 6-2.2 for details
Modbus	Message Router
Parameter	Message Router
Assembly	Message Router, I/O Connection

The CIP-Modbus Translator is a collaboration of objects that presents a CIP style interface to client process entities. Consequently the endianess of any data presented at these interfaces must comply with CIP conventions.

There are two principal cases to consider, ie Modbus Registers and Modbus bit data packed in an octet string.

A Modbus register is a 16-bit word. The actual data type of the value contained is unknown.

Modbus bit data is packed into octets for network presentation.

6-3 I/O Assembly Instances

Chapter 5-3 defines the ranges of Assembly Instance IDs that are used to access Modbus Tables in a Modbus Device. For I/O Connections this range is further limited to Instance IDs for Holding Registers and Input Registers.

Other objects with Instance IDs out of the range (0x00000001 .. 0x00040000) can be defined for other data types such as configuration assemblies, device specific parameters not mapped to Modbus tables, etc.

Since the object model in this section represents a generic Modbus Device, the Assemblies presented represent the range of possibilities rather than a device specific set of Assembly objects.

6-4 I/O Assembly Data Attribute Format

The number of I/O assembly instances, their instance IDs, and the size of their data attributes are specific to each Modbus Device. However, the data attribute, ie Attribute 3, for a given Assembly Instance ID always refers to the same place in its respective Modbus table. For example, an Assembly with InstanceID = 1 always refers to the first 125 Modbus registers in the holding register table independent of whether or not the device actually implements this part of the holding register table.

The Instance ID ranges for Assembly Objects mapped to Modbus Registers are designed to accommodate the maximum number of Modbus registers accessed by certain Modbus Function Codes. For example, the Modbus Function Code 03 Read Holding Registers can read from 1 to 125 Modbus registers (250 bytes max) from the target device. However this maximum may be further constrained in other Modbus Function Codes. For example, the Modbus Function Code 16 Write Multiple Registers can write from 1 to 123 Modbus registers (246 bytes max). These differences must be taken into account when using and/or configuring a device to use these Assembly object mappings.

Table 6-4.1 Modbus Assembly Data Mapping (Instance ID 0x00000001 to 0x00010000)

(Modbus Register Mapping in CIP little-endian order)

Instance ID	Byte	Meaning
0x00000001	0	Modbus holding register 1: LSB
	1	Modbus holding register 1: MSB
	...	
	248	Modbus holding register 125: LSB
	249	Modbus holding register 125: MSB
0x00000002	0	Modbus holding register 2: LSB
	1	Modbus holding register 2: MSB
	...	
	248	Modbus holding register 126: LSB
	249	Modbus holding register 126: MSB
...		
0x0000FFFF	0	Modbus holding register 65535: LSB
	1	Modbus holding register 65535: MSB
	2	Modbus holding register 65536: LSB
	3	Modbus holding register 65536: MSB
0x00010000	0	Modbus holding register 65536: LSB
	1	Modbus holding register 65536: MSB

Table 6-4.2 Modbus Assembly Data Meaning (Instance ID 0x00020001 to 0x00030000)

Instance ID	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x00020001	0	Coil 8	Coil 7	Coil 6	Coil 5	Coil 4	Coil 3	Coil 2	Coil 1
	1	Coil 16	Coil 15	Coil 14	Coil 13	Coil 12	Coil 11	Coil 10	Coil 9
	...								
	249	Coil 2000	Coil 1999	Coil 1998	Coil 1997	Coil 1996	Coil 1995	Coil 1994	Coil 1993
0x00020002	0	Coil 9	Coil 8	Coil 7	Coil 6	Coil 5	Coil 4	Coil 3	Coil 2
	1	Coil 17	Coil 16	Coil 15	Coil 14	Coil 13	Coil 12	Coil 11	Coil 10
	...								
	249	Coil 2001	Coil 2000	Coil 1999	Coil 1998	Coil 1997	Coil 1996	Coil 1995	Coil 1994
...									
0x0002FFFF	0	0	0	0	0	0	0	Coil 65536	Coil 65535
0x00030000	0	0	0	0	0	0	0	0	Coil 65536

The mappings of Assembly instances for Instance IDs 0x00010001 to 0x00020000 Input Registers and 0x00030001 to 0x00040000 Discrete are analogous to the previous tables, and are not separately described here.

Let us consider an example of a specific Modbus device. Let a Modbus Device have the following Modbus tables:

- Input registers 5, 6, 7
- Input registers 15, 16
- Coils 6, 7, 8, 9, 10, 11, 12, 13, 14
- Coils 51, 52, 53

This Modbus device would then be seen as a CIP device with the following Assembly instances:

Table 6-4.3 Assembly Data and Instances of a Specific Modbus Device

(Modbus Register Mapping in CIP little-endian order)

Instance ID	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x00010005	0	Modbus Input register 5: LSB							
	1	Modbus Input register 5: MSB							
	2	Modbus Input register 6: LSB							
	3	Modbus Input register 6: MSB							
	4	Modbus Input register 7: LSB							
	5	Modbus Input register 7: MSB							
0x00010006	0	Modbus Input register 6: LSB							
	1	Modbus Input register 6: MSB							
	2	Modbus Input register 7: LSB							
	3	Modbus Input register 7: MSB							
0x00010007	0	Modbus Input register 7: LSB							
	1	Modbus Input register 7: MSB							
0x0001000F	0	Modbus Input register 15: LSB							
	1	Modbus Input register 15: MSB							
	2	Modbus Input register 16: LSB							
	3	Modbus Input register 16: MSB							
0x00010010	0	Modbus Input register 16: LSB							
	1	Modbus Input register 16: MSB							
0x00020006	0	Coil 13	Coil 12	Coil 11	Coil 10	Coil 9	Coil 8	Coil 7	Coil 6
	1	0	0	0	0	0	0	0	Coil 14
0x00020007	0	Coil 14	Coil 13	Coil 12	Coil 11	Coil 10	Coil 9	Coil 8	Coil 7
0x00020008	0	0	Coil 14	Coil 13	Coil 12	Coil 11	Coil 10	Coil 9	Coil 8
0x00020009	0	0	0	Coil 14	Coil 13	Coil 12	Coil 11	Coil 10	Coil 9
0x0002000A	0	0	0	0	Coil 14	Coil 13	Coil 12	Coil 11	Coil 10
0x0002000B	0	0	0	0	0	Coil 14	Coil 13	Coil 12	Coil 11
0x0002000C	0	0	0	0	0	0	Coil 14	Coil 13	Coil 12
0x0002000D	0	0	0	0	0	0	0	Coil 14	Coil 13

Instance ID	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x0002000E	0	0	0	0	0	0	0	0	Coil 14
0x00020033	0	0	0	0	0	0	Coil 53	Coil 52	Coil 51
0x00020034	0	0	0	0	0	0	0	Coil 53	Coil 52
0x00020035	0	0	0	0	0	0	0	0	Coil 53

Note that even if the Modbus device has more than 125 contiguous Modbus registers, the data attribute of the assembly will never have a size bigger than 250 bytes.

6-5 Mapping I/O Assembly Data Attribute Components

The Assembly object Instance IDs defined in Chapter 5-3 correspond to Modbus tables of the target Modbus device. This mapping is managed by the Modbus translator function.

6-5.1 Implicit Connection Example

Assume that one wants to write implicitly 6 contiguous Modbus holding registers starting at Modbus register 16. No data is being returned. He will therefore configure a new I/O connection by providing the following parameters used by the originator to build the forward open request to the target.

- O-> T connection size = 18 (to transport 6 Modbus Holding Registers = 2 bytes CIP class 1 sequence count + 4 bytes 32-bit real time header + 12 bytes application data)
- O-> T connection path:
    - Class ID = 0x04 (Assembly Class ID)
    - Instance ID = 0x000000010 (Modbus Holding Register 16)
- T->O Connection size is 2 (2 bytes CIP class 1 sequence count)
- Class ID = 0x04 (Assembly Class ID)
- Instance ID = 0x000000010 (Modbus Holding Register 16)

Note that the use of the CIP Real Time Header for Class 1 implicit connections to Modbus Devices follows the following conventions. (See Volume 1, Chapter 3-6.1.1 and Chapter 3-6.1.4 for class 1 packet formats)

- All Modbus devices are considered to have a 32 bit run/idle header in the O->T direction
- All Modbus devices are considered to be modeless in the T->O direction.
- The translating function removes the header in the O->T direction.
- When IDLE is detected by the translating function, the O->T data transmission stops. T->O transmission continues.
- All Modbus EDS files shall indicate this header information.

6-5.2 Explicit Connection Example

If the translator supports the Get_Member and/or Set_Member Assembly services, one or several contiguous elements from the Modbus Tables can be accessed explicitly using this translation.

For example, assume that one wants to read 4 contiguous input registers starting at index 11. It will be achieved by:

- calling the service “Get_Member”,

- for Class ID = 0x04 (Assembly Class ID),
- Instance ID = 0x00010000B,
- with the parameter Member ID = 1, and the ExtendedProtocolID is set to 'Multiple Sequential Members', the number of members = 4,
- acknowledging that the Member ID = 1 was defined with a length of 16 bits.

6-6 Defining Device Configuration

It would be efficient to be able to configure Modbus Devices using configuration Assemblies.

However, there are no specifications that define requirements for how Modbus Devices shall present their configuration data to the network.

Consequently this section will not describe any specific conventions for configuring Modbus Devices.

Volume 7: Integration of Modbus Devices into the CIP Architecture

Chapter 7: EDS Files

Contents

7-1 Introduction

This chapter of the Modbus specification contains additions to the definition of electronic data sheets (EDS) that are Modbus specific. In addition, EDS examples are provided to show how a Modbus device may be modeled with an EDS file. See the Volume 1 for more information about the format of electronic data sheets and the definition of EDS related terms such as EDS section, EDS entry and EDS field. Although EDS files are not required for Modbus devices, providing an EDS file will improve the user experience in accessing the device from CIP originators.

The types of Modbus Device EDS files described in this chapter include:

- Device specific EDS files created by the manufacturer of a Modbus device. Complete example EDS files to be used as templates to facilitate the construction of a Modbus device specific EDS file are included.
- Generic Modbus Device EDS file to facilitate the configuration of Modbus target devices with no specific EDS file.

EDS files for Modbus devices shall be constructed using the EDS file conventions described in Volume 1. Additional keywords specific to Modbus devices are provided in this chapter.

The Generic Modbus Device EDS file defines the virtual CIP capability potentially present in any Modbus target device, allowing EDS-based CIP originators to communicate with the Modbus device. The Generic Modbus Device EDS specifies the complete Assembly and Parameter Instance ID range for Modbus holding registers and input registers, regardless of whether or not the device supports all of the instances. Using the Generic Modbus Device EDS, a user is required to know which specific Assembly or Parameter instance numbers are supported (based on the input and output register support of the Modbus device) when configuring communication with the target device.

A Modbus device may alternatively provide an EDS file specific to the device, allowing better usability with an EDS-based CIP originator. The device specific EDS file specifies the exact Assembly instances, Parameter instances, device identity, and communication parameters supported by the device. A device specific EDS file provides the user with a more refined view of which data items the device supports.

7-2 EDS File Support for Modbus Devices

The purpose of a Modbus EDS file is to provide CIP configuration tools and connection originators with information to identify Modbus devices and facilitate the creation of configuration data for a Modbus target device by a configuration tool. A device specific Modbus EDS file explicitly describes the data provided by the device and how this data is presented to the network. In addition, the device specific Modbus EDS file includes device identification information.

7-2.1 EDS Requirements

This section describes requirements for EDS file representing Modbus devices. These EDS files shall follow the requirements of Volume 1 Chapter 7, in addition to the requirements defined in this section. In order to provide a device specific EDS file, a manufacturer must have a valid Vendor ID from ODVA.

7-2.1.1 Device Description Section

The following items identify required field values for required entry keywords in the [Device] section of a device specific EDS file:

- The VendCode keyword shall contain the Vendor ID of the manufacturer
- The ProdType keyword shall be set to 28_{hex}
- The ProdTypeStr keyword shall be set to "Modbus Device"
- The ProdCode keyword shall be assigned a vendor specific value that is unique to the type of device being represented by the EDS file
- All other required keywords shall be present

Also, the following Modbus specific keyword is conditionally required:

Entry Name	Entry Keyword	Number of Fields	Data Type	Required/Optional
Modbus Identity Info	ModbusIdentityInfo	3	n/a	Conditional[1]

1 This entry keyword is required if the Modbus device supports the Read Device Identification function. Otherwise, this entry is not allowed.

Modbus Identity Info – This field provides the ASCII text for three of the Device Identification fields returned by the Modbus device (represented by this EDS file) within Attribute 18 (Modbus Identity Info) of the Identity Object. These values originate from the Read Device Identification function (Function Code 0x2B / MEI 0x0E) within the Modbus device. A configuration tool uses this entry to match the EDS to the device being configured. These three fields are defined below.

Field Name	Field Number	Data Type	Required/Optional
Vendor Name	1	ASCII Character Array	Required
Product Code	2	ASCII Character Array	Required
Major Minor Revision	3	ASCII Character Array	Required

- Vendor Name – The value the device returns in the VendorName field
- Product Code – The value the device returns in the ProductCode field
- Major Minor Revision – The value the device returns in the MajorMinorRevision field

7-2.1.2 Object Class Sections

The following class section is defined for Modbus devices:

CIP Object	Section Keyword Name	Required/Optional	See section
Modbus Object (44hex)	[Modbus Class]	Required	5-6

7-2.1.2.1 Modbus Class Section

The Modbus Class section begins with the keyword [Modbus Class]. The purpose of this section is to:

- Identify which Instance services are supported by the Modbus device.

The following table identifies the required entry keywords beyond those specified in Volume 1, Chapter 7 for the Modbus Class section:

Entry Name	Entry Keyword	Required/Optional
Instance Service Support	Instance_Services	Required

At a minimum, every Modbus device supports the Modbus Passthrough service (0x51) of this object. In addition, the EDS shall indicate the other services supported by the device. The Modbus Object is defined in Chapter 5 of this specification.

7-2.1.3 Device Classification

There are two reserved values for the ClassN keyword of the Device Classification section to denote a Modbus device. A device specific Modbus EDS file shall indicate at least one of the following two classifications:

- ModbusTCP
- ModbusSL

ModbusTCP indicates the device as having Modbus/TCP capabilities. ModbusSL indicates a Modbus "Serial Line" device having either Modbus/RTU and/or Modbus/ASCII capabilities.

7-2.1.4 Parameters Section

The Params section is used to provide the following:

- A description of individual Modbus data items
- Internal configuration choices, such as the RPI value used in a connection

If a Modbus data item is not individually accessible, but can be accessed as part of an assembly, it can be represented by a Param entry with an empty Link Path field.

7-2.1.5 Assembly Section

The Assembly section contains sets of contiguous data registers that can be connected to through I/O connections, used for the setting of configuration via the Data Segment in a Forward_Open during connection origination or accessed using explicit messaging (either connected or unconnected). These assemblies are built from parameters representing Modbus data items as described in the Params section.

Assemblies used for I/O connections can contain parameters referencing contiguous data items in the Holding Registers and Input Registers only. Assemblies used for configuration using a Data Segment during connection origination can contain parameters referencing contiguous data items in the Holding Registers only. Assemblies accessed via explicit messaging can contain parameters referencing contiguous data items in any of the Modbus data tables.

All members of an assembly must be accessible with Modbus function code reads and/or writes. If an assembly contains any data items that are not valid Modbus data items, accessing that assembly will fail.

7-2.1.6 Connection Manager Section

The Connection Manager section indicates which assemblies can be connected to using I/O messaging. Different Connection entries can be used with different assemblies to specify a variety of data sets (for example, only 1 holding register instead of potentially all available), or to specify an input only connection instead of a bi-directional connection.

In the ConnectionN Entry, the Connection Parameters field specifies the types of real time headers used by the O->T and the T->O implicit connections. The values used shall be consistent with the following conventions for the CIP Real Time formats for implicit connections to Modbus devices. (See Chapter 3 of Volume 1, the CIP Specification, for Real Time formats.)

- All Modbus devices are considered to have a 32-bit run/idle header in the O->T direction
- All Modbus devices are considered to be modeless in the T->O direction.

7-2.2 Device Specific EDS Examples

The sub-sections that follow show various examples of possible Modbus devices, and how the EDS can be used to represent them to CIP clients (configuration tools and connection originators). At the end of this chapter, complete EDS files are presented for each of these examples.

7-2.2.1 Example A: EDS for a Combination I/O Block – 16 bit Discrete Input & 16 bit Discrete Output

Scenario Description

In this example, a Modbus device supports both Discrete Input and Discrete Output data. The Modbus server in the target device supports 2 Modbus Function Codes for data access, Function Code 3 (Read Holding Registers) and Function Code 16 (Write Multiple Registers).

See section 7-4.1 for the EDS Text.

Application Data:

- 16 bits of Discrete Input
- 16 bits of Discrete Output

Network Presentation:

- When accessed, 16 bits of Discrete Input are packed into 16-bit Modbus Holding Register 1 and are read by Modbus Function Code 3, Read Holding Registers Service.
- When accessed, 16 bits of Discrete Output are packed into 16-bit Modbus Holding Register 2 and are written by Modbus Function Code 16, Write Multiple Registers Service.

EDS Strategy

Two parameters are defined for the I/O data, one for each of the holding registers. Param1 references the input register at Holding Register 1, and Param2 references the output at Holding Register 2.

Two Assembly Objects are also defined where one Assembly Object (Instance ID 0x00000001) contains Param1 as the single member, and the other Assembly Object (Instance ID 0x00000002) contains Param2 as the single member.

The Connection Manager section contains two different connections that are available. The first provides both inputs and outputs, while the second is an input only connection.

The [Modbus Class] section acknowledges the device's support for Modbus Function Codes 3 and 16 via the Modbus Object Read Holding Registers (0x4E) and Write Holding Registers (0x50) services.

7-2.2.2 Example B: EDS for a Modbus Device with Modbus Registers Having Multiple Semantics

Example Description

In this example a Modbus device supports both Discrete Input and Discrete Output data. The Modbus server in the target device only supports 2 Modbus Function Codes for data access, Function Code 3 (Read Holding Registers) and Function Code 16 (Write Multiple Registers). However both the Discrete Input data and the Discrete Output data are accessed at the same Modbus Register Index.

See section 7-4.2 for the EDS Text.

Application Data:

- 16 bits of Discrete Input
- 16 bits of Discrete Output

Network Presentation:

- When accessed, 16 bits of Discrete Input data are packed into 16-bit Modbus holding register 1 and are read by Modbus Function Code 3, Read Holding Registers Service.
- When accessed, 16 bits of Discrete Output data are also packed into 16-bit Modbus holding register 1 and are written by Modbus Function Code 16, Write Multiple Registers Service.

EDS Strategy

Two parameters are defined for the I/O data, one for each of the holding registers. Param30000 references the input register at Holding Register 1, and Param300001 references the output at Holding Register 2. Using the descriptor bits, Param300000 is indicated as Read Only and Param300001 as Write Only.

One Assembly Object is used that maps to Modbus Holding Register 1 through Instance ID 0x00000001. This one Assembly is specified for both the O->T and T->O implicit connections.

In this example the Assembly is not specifically defined. Rather, Parameter Objects are defined in such a way that their values can be used to define Implicit Connections to the related Assembly.

The [Modbus Class] section acknowledges the device's support for Modbus Function Codes 3 and 16.

7-2.2.3 Example C: EDS for a Modbus Device Having Scattered Configuration Parameters

Example Description

In this example a Modbus device provides configuration parameters that are mapped to Modbus Holding registers. However, the set of configuration parameters are not mapped to a contiguous block of Modbus Holding Registers.

See section 7-4.3 for the EDS Text.

Application Data:

- CfgParam1 - 16-bit unsigned integer
- CfgParam2 - 16-bit unsigned integer
- CfgParam3 - 16-bit unsigned integer
- 16 bits of Discrete Input
- 32 bits of Discrete Output

Network Presentation:

- CfgParam1 is mapped to Modbus holding register 101 and can be written using the Modbus Function Code 16 Write Multiple Registers service and read using the Modbus Function Code 3 Read Holding Registers service.
- CfgParam2 is mapped to Modbus holding register 201 and can be written using the Modbus Function Code 16 Write Multiple Registers service and read using the Modbus Function Code 3 Read Holding Registers service.
- CfgParam3 is mapped to Modbus holding register 301 and can be written using the Modbus Function Code 16 Write Multiple Registers service and read using the Modbus Function Code 3 Read Holding Registers service.
- When accessed, 16 bits of Discrete Input data are packed into Modbus Input Register 1, and are read by Modbus Function Code 4, Read Input Registers Service.

- When accessed, 32 bits of Discrete Output data are packed into Modbus Holding Registers 1 and 2, and are written by Modbus Function Code 16, Write Multiple Registers Service

EDS Strategy

Each of the configuration parameters is represented as a parameter in the Params section. Since the parameters are not contiguous, an assembly can not be constructed to represent them as a single set. The input and output registers are also represented as parameters and included in assemblies, which are used in the I/O connection.

The [Modbus Class] section acknowledges the device's support for Modbus Function Codes 3, 4, and 16.

7-2.2.4 Example D: EDS for a Modbus Device Having a Contiguous Block of Configuration Parameters

Example Description

This example is a modification on the Modbus device presented in Example C. Instead of the configuration parameters being scattered in the Holding Registers table, they are contiguous. Thus, the parameters can be modeled by, and access as, an assembly. This assembly can be written to using explicit messaging, or included in the connection origination process using the Data Segment within a Forward_Open.

See section 7-4.4 for the EDS Text.

Application Data:

- CfgParam1 - 16-bit unsigned integer
- CfgParam2 - 16-bit unsigned integer
- CfgParam3 - 16-bit unsigned integer
- 16 bits of Discrete Input
- 32 bits of Discrete Output

Network Presentation:

- CfgParam1 is mapped to Modbus holding register 101 and can be written using the Modbus Function Code 16 Write Multiple Registers service and read using the Modbus Function Code 3 Read Holding Registers service.
- CfgParam2 is mapped to Modbus holding register 102 and can be written using the Modbus Function Code 16 Write Multiple Registers service and read using the Modbus Function Code 3 Read Holding Registers service.
- CfgParam3 is mapped to Modbus holding register 103 and can be written using the Modbus Function Code 16 Write Multiple Registers service and read using the Modbus Function Code 3 Read Holding Registers service.
- When accessed, 16 bits of Discrete Input data are packed into Modbus Input Register 1, and are read by Modbus Function Code 4, Read Input Registers Service.
- When accessed, 32 bits of Discrete Output data are packed into Modbus Holding Registers 1 and 2, and are written by Modbus Function Code 16, Write Multiple Registers Service

EDS Strategy

Since the configuration parameters are mapped to a contiguous block of Modbus Holding Registers, they can be mapped to a configuration assembly. The Forward_Open service can also be used to transport the device's configuration assembly in a Data Segment.

The [Modbus Class] section acknowledges the device's support for Modbus Function Codes 3, 4, and 16.

7-2.2.5 Example E: EDS for a Modbus/TCP Serial Gateway

Example Description

A Modbus/TCP Serial Gateway is a device with at least one Modbus/TCP port and one Modbus Serial port. This device can route messages from Modbus/TCP to Modbus Serial. The gateway may also have valid data items for reading/writing (including input and output data). In this example, there is one configuration register and one status register available

See section 7-4.4 for the EDS Text.

Application Data:

- CfgParam - 16-bit unsigned integer
- StatusParam - 16-bit unsigned integer

Network Presentation:

- CfgParam is mapped to Modbus holding register 1 and can be written using the Modbus Function Code 16 Write Multiple Registers service and read using the Modbus Function Code 3 Read Holding Registers service.
- StatusParam is mapped to Modbus holding register 2 and can be read using the Modbus Function Code 3 Read Holding Registers service.

EDS Strategy

The Port section of the EDS is used to indicate the presence of additional Modbus ports to the configuration tool. In this example, the device declares two ports in the Port section, one for Modbus/TCP and one for Modbus Serial. The EDS also defines the configuration and status parameters available.

The [Modbus Class] section acknowledges the device's support for Modbus Function Codes 3 and 16.

7-3 Generic EDS File for Modbus

The purpose of the Generic Modbus Device EDS file is to facilitate an interactive I/O connection setup for a Modbus target device by a configuration tool when a device specific EDS file is not available for a Modbus device. Using this EDS file with the configuration tool, the user will provide additional information from the Modbus device data sheet and/or user manual.

The Generic Modbus Device EDS file provides a generalized view of the input and output data of a Modbus device. This EDS file also provides the values of Identity components reserved for Modbus devices such as the VendorCode, ProductType, and ProdTypeStr.

It is intended that CIP configuration tools will include this Generic Modbus Device EDS file in their product, and allow the user to add any Modbus device to the network configuration using this EDS. Only one generic Modbus EDS file shall be provided by a configuration tool.

7-3.1 Generic Modbus Device EDS File Content

Text for the Generic Modbus EDS File is found in section 7-5. This file is available for download from the ODVA website (www.odva.org).

7-3.1.1 Device Description Section

There is a reserved value for VendCode entry keyword in the [Device] section of the EDS that signifies a Modbus Device without specifying the actual manufacturer. This value is 65534, as shown below:

VendCode = 65534;

Also there are reserved values for ProdType and ProdTypeStr to signify Modbus devices. They are:

```
ProdType = 40;                  $ This Product (Device) Type code for
                                $ Modbus devices
ProdTypeStr = "Modbus Device"; $ This Product (Device) Type String for
                                $ Modbus devices
```

7-3.1.2 Object Class Sections

7-3.1.2.1 Modbus Class Section

The Generic Modbus EDS defines all instance services as valid. This will indicate that the client should allow all services, even though some may fail due to lack of support by the target device.

7-3.1.3 Device Classification Section

To indicate a Modbus device, keyword values of both ‘ModbusTCP’ and ‘ModbusSL’ are specified. This will allow the device to be added to either type of Modbus network.

```
[Device Classification]
                            Class1 = “ModbusTCP”;
                            Class2 = “ModbusSL”;
```

This will allow the device to appear on either type of network.

7-3.1.4 Parameters Section

Generally a Modbus device presents some input and/or output data to the network. This section parameterizes the input and output data of a generic Modbus device.

The input data is parameterized by:

- Number of registers (Param300001)
- Starting Modbus Register (either Param300003 from input registers or Param300007 from Holding Registers
- Input Cyclic Interval (Param300005)

The output data is parameterized by:

- Number of Output registers (Param300002)
- Starting Modbus Output Register (Param300004)
- Output Cyclic Interval (Param300006)

Device Configuration Data Parameters are not modeled because there is no defined convention for this in Modbus devices.

7-3.1.5 Assembly Section

No Assemblies are specifically defined in the [Assembly] section of the Generic Modbus Device EDS file. Rather, ConnectionN entries are defined, each as a connection to an Assembly whose definition is built through parameterization. This definition intends to accommodate the range of possible definitions.

7-3.1.6 Connection Manager Section

The Generic Modbus Device EDS defines several ConnectionN entries. These entries are all bi-directional and specify either data transfer in both directions, or O->T (output) data with a heartbeat connection (no input data) in the T->O direction. The input/output data is represented as an Assembly Object with the Instance ID set to the index of the first register of the input /output block as follows:

- O ->T is to the first register index of the output data (Param300004).
- T ->O is from the first register index of the input data (Param300003).

The O->T and T->O connections sizes are represented in Param300002 and Param300001 respectively. Both Parameters are defined as number of bytes, with divisors set to a value of 2 so that the Param value displayed can be shown as number of 16 bit registers.

7-4 EDS File Examples

This section contains the complete text representations of the EDS examples described earlier in this chapter. Places in these examples that contain the following comment: <TODO: …> indicates an area that needs to be modified by the device manufacturer when using these examples to construct an actual EDS for their product.

7-4.1 Text for Example A: Combination I/O Block – 16 bits Discrete Input & 16 bits Discrete Output

```
$ Example Description
$ In this example a Modbus Device supports both Discrete Input and Discrete Output
$ data.  The Modbus server at the device supports 2 Modbus Function Codes for data
$ access,  Function Code 3 (Read Holding Registers) and Function Code 16 (Write
$ Multiple Registers).
$
$ Application Data:
$  • 16 bits of Discrete Input
$  • 16 bits of Discrete Output
$
$ Network Presentation:
$  • When accessed, 16 bits of Discrete Input are packed into 16-bit Modbus holding
$    register 1 and are read by Modbus Function Code 3, Read Holding Registers.
$  • When accessed, 16 bits of Discrete Output are also packed into 16-bit Modbus
$    holding register 2 and are written by Modbus Function Code 16, Write Multiple
$    Registers.
$
$ EDS Strategy
$ The basic strategy for this EDS file is to define the constructs that accommodate
$ the network presentation of the data and then to model the application data
$ within these constructs.
$
$ Two Assembly Objects are defined where one Assembly Object maps to Modbus
$ Holding Register 0x0001 through InstanceID 0x00000001 and the other Assembly
$ Object maps to Modbus Holding Register 0x0002 through InstanceID 0x00000002.

[File]
  DescText = "Combo Device = 16 Discrete Inputs + 16 Discrete Outputs";
  $ <TODO: Update Dates, Times, and Revision>
  CreateDate = ; $ e.g.: 04-04-2007
  CreateTime = ; $ e.g.: 12:00:00
  ModDate = ; $ e.g.: 09-01-2007
  ModTime = ; $ e.g.: 05:05:00
  Revision = 1.2;
  HomeURL = "<TODO: Fill in location of EDS file here, or remove this keyword>";

[Device]
  VendCode = ; $ <TODO: Fill in your VendorID number here> $ Vendor Number
  VendName = "<TODO: Fill in your vendor name here>";      $ Vendor Name
  ProdType = 40 ;                  $ Device Type code for Modbus devices
  ProdTypeStr = "Modbus Device";   $ Device Type string for Modbus devices
  ProdCode = ; $ <TODO: Fill in this product's code value here>
  MajRev = ; $ <TODO: Fill in this product's major revision value here>
  MinRev = 1;
  ProdName = "<TODO: Fill in your device product name string here>";
  Catalog = "<TODO: Fill in your catalog string here>";
  Icon = "<TODO: Fill in your icon filename string here>";

  $ Include the ModusIdentityInfo keyword only if your device supports the Read
  $ Device Identification function. If supported, the values entered below shall match
  $ the values returned from that function.
  ModbusIdentityInfo = "TODO: Fill in Vendor name string",
                       "TODO: Fill in Product Code string",
                       "TODO: Fill in Product revision string";
```

```
$ The device classification value, ModbusTCP or ModbusSL
[Device Classification]
   Class1 = ; $ <TODO: Fill in ModbusTCP or ModbusSL here>

[Params]
  $ <TODO: Create input and output parameters specific to your device here>
  $ Param1 represents the 16 Discrete Inputs at Holding Register 1
  $ Used as Parameter Value attribute, it allows the definition of EDS Enumerations
  $ that map to the 16 Discrete Inputs of the application data.

  Param1 =
    0,                          $ reserved, shall equal 0
    ,"",                        $ Link Path Size, Link Path
    0x0000,                     $ Descriptor
    0xD2,                       $ Data Type
    2,                          $ Data Size in bytes
    "Discrete Inputs",          $ name
    "",                         $ units
    "16 Discrete Inputs at Holding Register 1",    $ help string
    0x0001,0x8000,0x0000,       $ min, max, default data values
    ,,,,                        $ mult, div, base, offset scaling
    ,,,,                        $ mult, div, base, offset links
    ;                           $ decimal places
  Enum1 =
    0,"DiscreteInput1",
    1,"DiscreteInput2",
    2,"DiscreteInput3",
    3,"DiscreteInput4",
    4,"DiscreteInput5",
    5,"DiscreteInput6",
    6,"DiscreteInput7",
    7,"DiscreteInput8",
    8,"DiscreteInput9",
    9,"DiscreteInput10",
    10,"DiscreteInput11",
    11,"DiscreteInput12",
    12,"DiscreteInput13",
    13,"DiscreteInput14",
    14,"DiscreteInput15",
    15,"DiscreteInput16";
```

```
$ Param2 represents the 16 Discrete Outputs at Holding Register 2
$ Used as Parameter Value attribute, it allows the definition of EDS Enumerations
$ that map to the 16 Discrete Outputs of the application data.
Param2 =
  0,                          $ reserved, shall equal 0
  ,"",                        $ Link Path Size, Link Path
  0x0000,                     $ Descriptor
  0xD2,                       $ Data Type
  2,                          $ Data Size in bytes
  "Discrete Outputs",         $ name
  "",                         $ units
  "16 Discrete Outputs at Holding Register 2",    $ help string
  0x0001,0x8000,0x0000,       $ min, max, default data values
  ,,,,                        $ mult, div, base, offset scaling
  ,,,,                        $ mult, div, base, offset links
  ;                           $ decimal places
Enum2 =
  0,"DiscreteOutput1",
  1,"DiscreteOutput2",
  2,"DiscreteOutput3",
  3,"DiscreteOutput4",
  4,"DiscreteOutput5",
  5,"DiscreteOutput6",
  6,"DiscreteOutput7",
  7,"DiscreteOutput8",
  8,"DiscreteOutput9",
  9,"DiscreteOutput10",
  10,"DiscreteOutput11",
  11,"DiscreteOutput12",
  12,"DiscreteOutput13",
  13,"DiscreteOutput14",
  14,"DiscreteOutput15",
  15,"DiscreteOutput16";

Param300005 =
  0,                          $ reserved, shall equal 0
  ,,                          $ Link Path Size, Link Path
  0x0000,                     $ Descriptor
  0xC8,                       $ Data Type
  4,                          $ Data Size in bytes
  "Input Cyclic Interval",    $ name
  "ms",                       $ units
  "",                         $ help string
  10000,10000000,10000,       $ min, max, default data values
  ,,,,                        $ mult, div, base, offset scaling
  ,,,,                        $ mult, div, base, offset links
  ;                           $ decimal places
```

```
  Param300006 =
    0,                          $ reserved, shall equal 0
    ,,                          $ Link Path Size, Link Path
    0x0000,                     $ Descriptor
    0xC8,                       $ Data Type
    4,                          $ Data Size in bytes
    "Output Cyclic Interval",   $ name
    "ms",                       $ units
    "",                         $ help string
    10000,10000000,10000,       $ min, max, default data values
    ,,,,                        $ mult, div, base, offset scaling
    ,,,,                        $ mult, div, base, offset links
    ;                           $ decimal places

[Assembly]       $ <TODO: Create assemblies specific to your device here>
  Assem1 = "Input Data",        $ Assembly name
    ,                           $ Path
    2,                          $ Assembly size
    ,,,
    , Param1;                   $ Input Parameter

  Assem2 = "Output Data",       $ Assembly name
    ,                           $ Path
    2,                          $ Assembly size
    ,,,
    , Param2;                   $ Output Parameter

[Connection Manager]
  $ Connection 1 is an exclusive only connection
  Connection1 =
    0x04010002,             $ Transport classes = 2
                            $ Production Trigger = Cyclic
                            $ Transport type = Exclusive-owner
                            $ Client
    0x44640405,             $ O->T fixed size supported
                            $ T->O fixed size supported
                            $ O->T Real time transfer format = 32 bit run/idle header
                            $ T->O Real time transfer format = Modeless
                            $ O->T connection type: POINT2POINT
                            $ T->O connection type: MULTICAST
                            $ T->O connection type: POINT2POINT
                            $ O->T priority: SCHEDULED
                            $ T->O priority: SCHEDULED
    Param300006, , Assem2,  $ O->T RPI, size, format
                            $ <TODO: Insert the O->T RPI, size and format information
                            $ for your connection here>
    Param300005, , Assem1,  $ T->O RPI, size, format
                            $ <TODO: Insert the T->O RPI, size and format information
                            $ for your connection here>
    ,,                      $ config #1 size, format
    ,,                      $ config #2 size, format
    "<TODO: Insert your connection name here>",     $ Connection Name
    "< TODO: Insert your connection help here >",   $ help string
    "20 04 2E 00 02 00 00 00 2E 00 01 00 00 00";    $ Path <TODO: Insert the
                            $ assembly numbers for your O->T and T->O data here>
```

```
  $ Connection 2 is an input only connection
  Connection2 =
    0x02010002,             $ Transport classes = 2
                            $ Production Trigger = Cyclic
                            $ Transport type = Input-only
                            $ Client

    0x44640405,             $ O->T fixed size supported
                            $ T->O fixed size supported
                            $ O->T Real time transfer format = 32 bit run/idle header
                            $ T->O Real time transfer format = Modeless
                            $ O->T connection type: POINT2POINT
                            $ T->O connection type: MULTICAST
                            $ T->O connection type: POINT2POINT
                            $ O->T priority: SCHEDULED
                            $ T->O priority: SCHEDULED
    ,0,,                    $ O->T RPI, size, format
    Param300005, 2, Assem2, $ T->O RPI, size, format
                            $ <TODO: Insert the T->O RPI, size and format information
                            $ for your connection here>
      ,,                    $ config #1 size, format
      ,,                    $ config #2 size, format
      "<TODO: Insert your connection name here>",   $ Connection Name
      "< TODO: Insert your connection help here >", $ help string
      "20 04 2E 00 00 00 00 00 2E 00 01 00 00 00";  $ Path <TODO: Insert the assembly
                            $ number for your T->O data here>

[Modbus Class]
   Revision = 1;
   MaxInst = 1;
   Number_Of_Static_Instances = 1;
   Max_Number_Of_Dynamic_Instances = 0;

  Instance_Services =
            0x4E,                   $ Modbus Function Code 3, Read Holding Registers
            0x50,                   $ Modbus Function Code 16, Write Holding Registers
            0x51;                   $ Modbus Passthrough service
```

7-4.2 Text for Example B: Modbus Registers Having Multiple Semantics

```
$ Example Description
$    In this example a Modbus Device supports both Discrete Input and Discrete Output
$   data.  The Modbus server at the device only supports 2 Modbus Function Codes for
$   data access, Function Code 3 (Read Holding Registers) and Function Code 16 (Write
$   Multiple Registers. However, both the Discrete Input data and the Discrete Output
$   data are accessed at the same Modbus Register Index.
$
$ Application Data:
$  • 16 bits of Discrete Input
$  • 16 bits of Discrete Output
$
$ Network Presentation:
$  • When accessed, 16 bits of Discrete Input are packed into 16-bit Modbus holding
$    register 1 and are read by Modbus Function Code 3, (Read Holding Registers).
$  • When accessed, 16 bits of Discrete Output are also packed into 16-bit Modbus
$    holding register 1 and are written by Modbus Function Code 16,
$    (Write Multiple Registers).
$
$ EDS Strategy
$  The basic strategy for this EDS file is to define the constructs that accommodate
$  the network presentation of the data and then to model the application data within
$  these constructs.
$
$  One Assembly Object is defined that maps to Modbus Holding Register 1 through
$  InstanceID 0x00000001. This one Assembly is used for both the O->T and T->O implicit
$  connections. In this example the Assembly is not specifically defined. Rather,
$  Parameter Objects are defined in such a way that their values can be used to define
$  Implicit Connections to the related Assembly.

[File]
  DescText = "Modbus Registers Multiple Semantics";
  $ <TODO: Update Dates, Times, and Revision>
  CreateDate = ; $ e.g.: 04-04-2007
  CreateTime = ; $ e.g.: 12:00:00
  ModDate = ; $ e.g.: 09-01-2007
  ModTime = ; $ e.g.: 05:05:00
  Revision = 1.2;
  HomeURL = "<TODO: Fill in location of EDS file here, or remove this keyword>";

[Device]
  VendCode = ; $ <TODO: Fill in your VendorID number here> $ Vendor Number
  VendName = "<TODO: Fill in your vendor name here>";      $ Vendor Name
  ProdType = 40;                   $ Device Type code for Modbus devices
  ProdTypeStr = "Modbus Device";   $ Device Type string for Modbus devices
  ProdCode = ; $ <TODO: Fill in this product's code value here>
  MajRev = ; $ <TODO: Fill in this product's major revision value here>
  MinRev = 1;
  ProdName = "<TODO: Fill in your device product name string here>";
  Catalog = "<TODO: Fill in your catalog string here>";
  Icon = "<TODO: Fill in your icon filename string here>";

  $ Include the ModusIdentityInfo keyword only if your device supports the Read
  $ Device Identification function. If supported, the values entered below shall match
```

```
  $ the values returned from that function.
  ModbusIdentityInfo = "TODO: Fill in Vendor name string",
                       "TODO: Fill in Product Code string",
                       "TODO: Fill in Product revision string";
$ The device classification value, ModbusTCP or ModbusSL
[Device Classification]
  Class1 = ; $ <TODO: Fill in ModbusTCP or ModbusSL here>

$ For ParamN Entries, N is defined as follows
$ 0x00000001 - 0x00010000     ( 1 .. 65536)       Holding Registers
$ 0x00010001 - 0x00020000     (65537 .. 131072)   Input Registers
$ 0x00020001 - 0x00030000     (131073 .. 196608)  Coils
$ 0x00030001 - 0x00040000     (196609 .. 262144)  Discrete Inputs
$ Parameter Instances with values of N > 262144 are not mapped to Modbus tables.

[Params]
  Param300000 =
    0,                          $ reserved, shall equal 0
    10, "20 0F 24 01 30 01",    $ Link Path Size, Link Path
    0x0010,                     $ Descriptor = Read Only
    0xD2,                       $ Data Type
    2,                          $ Data Size in bytes
    "Discrete Inputs",          $ Name
    "",                         $ units
    "16 Discrete Inputs at Holding Register 1",    $ help string
    0x0001,0x8000,0x0000,       $ min, max, default data values
    ,,,,                        $ mult, div, base, offset scaling
    ,,,,                        $ mult, div, base, offset links
    ;                           $ decimal places
  Enum300000 =
    0,"DiscreteInput1",
    1,"DiscreteInput2",
    2,"DiscreteInput3",
    3,"DiscreteInput4",
    4,"DiscreteInput5",
    5,"DiscreteInput6",
    6,"DiscreteInput7",
    7,"DiscreteInput8",
    8,"DiscreteInput9",
    9,"DiscreteInput10",
    10,"DiscreteInput11",
    11,"DiscreteInput12",
    12,"DiscreteInput13",
    13,"DiscreteInput14",
    14,"DiscreteInput15",
    15,"DiscreteInput16";

  Param300001 =
    0,                          $ reserved, shall equal 0
    10, "20 0F 24 01 30 01",    $ Link Path Size, Link Path
    0x4000,                     $ Descriptor = Write Only
    0xD2,                       $ Data Type
    2,                          $ Data Size in bytes
    "Discrete Outputs",         $ Name
    "",                         $ units
    "16 Discrete Outputs at Holding Register 1",    $ help string
```

```
    0x0001,0x8000,0x0000,      $ min, max, default data values
    ,,,,                       $ mult, div, base, offset scaling
    ,,,,                       $ mult, div, base, offset links
    ;                          $ decimal places
  Enum300001 =
    0,"DiscreteOutput1",
    1,"DiscreteOutput2",
    2,"DiscreteOutput3",
    3,"DiscreteOutput4",
    4,"DiscreteOutput5",
    5,"DiscreteOutput6",
    6,"DiscreteOutput7",
    7,"DiscreteOutput8",
    8,"DiscreteOutput9",
    9,"DiscreteOutput10",
    10,"DiscreteOutput11",
    11,"DiscreteOutput12",
    12,"DiscreteOutput13",
    13,"DiscreteOutput14",
    14,"DiscreteOutput15",
    15,"DiscreteOutput16";

  Param300005 =
    0,                         $ reserved, shall equal 0
    ,,                         $ Link Path Size, Link Path
    0x0000,                    $ Descriptor
    0xC8,                      $ Data Type
    4,                         $ Data Size in bytes
    "Input Cyclic Interval",   $ name
    "ms",                      $ units
    "Input Cyclic Interval",   $ help string
    10000,10000000,10000,      $ min, max, default data values
    ,,,,                       $ mult, div, base, offset scaling
    ,,,,                       $ mult, div, base, offset links
    0;                         $ decimal places
  Param300006 =
    0,                         $ reserved, shall equal 0
    ,,                         $ Link Path Size, Link Path
    0x0000,                    $ Descriptor
    0xC8,                      $ Data Type
    4,                         $ Data Size in bytes
    "Output Cyclic Interval",  $ name
    "ms",                      $ units
    "Output Cyclic Interval",  $ help string
    10000,10000000,10000,      $ min, max, default data values
    ,,,,                       $ mult, div, base, offset scaling
    ,,,,                       $ mult, div, base, offset links
    0;                         $ decimal places

[Connection Manager]
  $ For Connection 1
  $ O->T is a connection to Holding Register 1 for the Output Data
  $ T->O is a connection to Holding Register 1 for the Input Data.
  Connection1 =
    0x04010002,            $ Transport classes = 2
                           $ Production Trigger = Cyclic
```

```
                          $ Transport type = Exclusive-owner
                          $ Client

    0x44640405,           $ O->T fixed size supported
                          $ T->O fixed size supported
                          $ O->T Real time transfer format = 32 bit run/idle header
                          $ T->O Real time transfer format = Modeless
                          $ O->T connection type: POINT2POINT
                          $ T->O connection type: MULTICAST
                          $ T->O connection type: POINT2POINT
                          $ O->T priority: SCHEDULED
                          $ T->O priority: SCHEDULED
    Param300006, 2, Param300001,     $ O->T RPI, size, format
    Param300005, 2, Param300000,     $ T->O RPI, size, format
    ,,                               $ config #1 size, format
    ,,                               $ config #2 size, format
    "<TODO: Insert your connection name here>",      $ Connection Name
    "< TODO: Insert your connection help here >",    $ help string
    "20 04 2E 00 01 00 00 00 2E 00 01 00 00 00";     $ Path <TODO: Insert the
                                     $ assembly number for your T->O data here>

[Modbus Class]
    Revision = 1;
    MaxInst = 1;
    Number_Of_Static_Instances = 1;
    Max_Number_Of_Dynamic_Instances = 0;

    Instance_Services =
            0x4E,                  $ Modbus Function Code 3, Read Holding Registers
            0x50,                  $ Modbus Function Code 16 Write Holding Registers
            0x51;                  $ Modbus Passthrough service
```

7-4.3 Text for Example C: Scattered Configuration Parameters

```
$ Example Description
$ In this example a Modbus device provides configuration parameters that are mapped to
$ Modbus Holding registers. However, the set of configuration parameters are not mapped
$ to a contiguous block of Modbus Holding Registers.
$ Application Data:
$ - CfgParam1 - 16-bit unsigned integer
$ - CfgParam2 - 16-bit unsigned integer
$ - CfgParam3 - 16-bit unsigned integer
$ - 16 bits of Discrete Input
$ - 32 bits of Discrete Output
$ Network Presentation:
$ - CfgParam1 is mapped to Modbus holding register 101 and can be written using the
$   Modbus Function Code 16 Write Multiple Registers service and read using the Modbus
$   Function Code 3 Read Holding Registers service.
$ - CfgParam2 is mapped to Modbus holding register 201 and can be written using the
$   Modbus Function Code 16 Write Multiple Registers service and read using the Modbus
$   Function Code 3 Read Holding Registers service.
$ - CfgParam3 is mapped to Modbus holding register 301 and can be written using the
$   Modbus Function Code 16 Write Multiple Registers service and read using the Modbus
$   Function Code 3 Read Holding Registers service.
$ - When accessed, 16 bits of Discrete Input data are packed into Modbus Input Register
$   1, and are read by Modbus Function Code 4, Read Input Registers Service.
$ - When accessed, 32 bits of Discrete Output data are packed into Modbus Holding
$   Registers 1 and 2, and are written by Modbus Function Code 16, Write Multiple
$   Registers Service.
$
$ EDS Strategy
$ Each of the configuration parameters is represented as a parameter in the Params
$ section. Since the parameters are not contiguous, an assembly can not be constructed
$ to represent them as a single set. The input and output registers are also
$ represented as parameters and included in assemblies, which are used in the I/O
$ connection. The [Modbus Class] section acknowledges the device's support for Modbus
$ Function Codes 3, 4, and 16.

[File]
  DescText = "Scattered Configuration";
  $ <TODO: Update Dates, Times, and Revision>
  CreateDate = ; $ e.g.: 04-04-2007
  CreateTime = ; $ e.g.: 12:00:00
  ModDate = ; $ e.g.: 09-01-2007
  ModTime = ; $ e.g.: 05:05:00
  Revision = 1.2;
  HomeURL = "<TODO: Fill in location of EDS file here, or remove this keyword>";
```

```
[Device]
  VendCode = ; $ <TODO: Fill in your VendorID number here>  $ Vendor Number
  VendName = "<TODO: Fill in your vendor name here>";       $ Vendor Name
  ProdType = 40 ;                  $ Device Type code for Modbus devices
  ProdTypeStr = "Modbus Device";   $ Device Type string for Modbus devices
  ProdCode = ; $ <TODO: Fill in this product's code value here>
  MajRev = ; $ <TODO: Fill in this product's major revision value here>
  MinRev = 1;
  ProdName = "<TODO: Fill in your device product name string here>";
  Catalog = "<TODO: Fill in your catalog string here>";
  Icon = "<TODO: Fill in your icon filename string here>";
  $ Include the ModusIdentityInfo keyword only if your device supports the Read
  $ Device Identification function. If supported, the values entered below shall match
  $ the values returned from that function.
  ModbusIdentityInfo = "TODO: Fill in Vendor name string",
                       "TODO: Fill in Product Code string",
                       "TODO: Fill in Product revision string";

$ The device classification value, ModbusTCP or ModbusSL
[Device Classification]
  Class1 = ; $ <TODO: Fill in ModbusTCP or ModbusSL here>

$ For ParamN Entries, N is defined as follows
$ 0x00000001 - 0x00010000     ( 1 .. 65536)       Holding Registers
$ 0x00010001 - 0x00020000     (65537 .. 131072)   Input Registers
$ 0x00020001 - 0x00030000     (131073 .. 196608)  Coils
$ 0x00030001 - 0x00040000     (196609 .. 262144)  Discrete Inputs

[Params]
$ Param1 represents the 16 Discrete Outputs at Holding Register 1
$ Used as Parameter Value attribute, it allows the definition of EDS Enumerations that
$ map to the first 16 Discrete Outputs of the application data.
  Param1 =
    0,                          $ reserved, shall equal 0
    ,"",                        $ Link Path Size, Link Path
    0x0000,                     $ Descriptor
    0xD2,                       $ Data Type
    2,                          $ Data Size in bytes
    "Discrete Outputs 1 - 16",  $ name
    "",                         $ units
    "16 Discrete Outputs at Holding Register 1",    $ help string
    0x0001,0x8000,0x0000,       $ min, max, default data values
    ,,,,                        $ mult, div, base, offset scaling
    ,,,,                        $ mult, div, base, offset links
    ;                           $ decimal places
  Enum1 =
    0,"DiscreteOutput1",
    1,"DiscreteOutput2",
    2,"DiscreteOutput3",
    3,"DiscreteOutput4",
    4,"DiscreteOutput5",
    5,"DiscreteOutput6",
    6,"DiscreteOutput7",
    7,"DiscreteOutput8",
    8,"DiscreteOutput9",
    9,"DiscreteOutput10",
```

```
  10,"DiscreteOutput11",
  11,"DiscreteOutput12",
  12,"DiscreteOutput13",
  13,"DiscreteOutput14",
  14,"DiscreteOutput15",
  15,"DiscreteOutput16";

$ Param2 represents the 16 Discrete Outputs at Holding Register 2
$ Used as Parameter Value attribute, it allows the definition of EDS Enumerations that
$ map to the next 16 Discrete Outputs of the application data.
 Param2 =
  0,                          $ reserved, shall equal 0
  ,"",                        $ Link Path Size, Link Path
  0x0000,                     $ Descriptor
  0xD2,                       $ Data Type
  2,                          $ Data Size in bytes
  "Discrete Outputs 17 - 32", $ name
  "",                         $ units
  "16 Discrete Outputs at Holding Register 2",    $ help string
  0x0001,0x8000,0x0000,       $ min, max, default data values
  ,,,,                        $ mult, div, base, offset scaling
  ,,,,                        $ mult, div, base, offset links
  ;                           $ decimal places
 Enum2 =
  0,"DiscreteOutput17",
  1,"DiscreteOutput18",
  2,"DiscreteOutput19",
  3,"DiscreteOutput20",
  4,"DiscreteOutput21",
  5,"DiscreteOutput22",
  6,"DiscreteOutput23",
  7,"DiscreteOutput24",
  8,"DiscreteOutput25",
  9,"DiscreteOutput26",
  10,"DiscreteOutput27",
  11,"DiscreteOutput28",
  12,"DiscreteOutput29",
  13,"DiscreteOutput30",
  14,"DiscreteOutput31",
  15,"DiscreteOutput32";
 Param101 =
  0,                          $ reserved, shall equal 0
  , "",                       $ Link Path Size, Link Path
  0x0000,                     $ Descriptor
  0xC7,                       $ Data Type
  2,                          $ Data Size in bytes
  "CfgParam1",                $ name
  "",                         $ units
  "CfgParam1",                $ help string
  ,,0,                        $ min, max, default data values
  ,,,,                        $ mult, div, base, offset scaling
  ,,,,                        $ mult, div, base, offset links
  ;                           $ decimal places
 Param201 =
  0,                          $ reserved, shall equal 0
  , "",                       $ Link Path Size, Link Path
```

```
  0x0000,                     $ Descriptor
  0xC7,                       $ Data Type
  2,                          $ Data Size in bytes
  "CfgParam2",                $ name
  "",                         $ units
  "CfgParam2",                $ help string
  ,,0,                        $ min, max, default data values
  ,,,,                        $ mult, div, base, offset scaling
  ,,,,                        $ mult, div, base, offset links
  ;                           $ decimal places
Param301 =
  0,                          $ reserved, shall equal 0
  , "",                       $ Link Path Size, Link Path
  0x0010,                     $ Descriptor
  0xC7,                       $ Data Type
  2,                          $ Data Size in bytes
  "CfgParam3",                $ name
  "",                         $ units
  "CfgParam3",                $ help string
  ,,0,                        $ min, max, default data values
  ,,,,                        $ mult, div, base, offset scaling
  ,,,,                        $ mult, div, base, offset links
  ;                           $ decimal places
$ Param65537 represents the 16 Discrete Inputs at Input Register 1
$ Used as Parameter Value attribute, it allows the definition of EDS Enumerations that
$ map to the 16 Discrete Inputs of the application data.

Param65537 =
  0,                          $ reserved, shall equal 0
  ,"",                        $ Link Path Size, Link Path
  0x0010,                     $ Descriptor = Read Only
  0xD2,                       $ Data Type
  2,                          $ Data Size in bytes
  "Discrete Inputs",          $ name
  "",                         $ units
  "16 Discrete Inputs at Input Register 1",    $ help string
  0x0001,0x8000,0x0000,       $ min, max, default data values
  ,,,,                        $ mult, div, base, offset scaling
  ,,,,                        $ mult, div, base, offset links
  ;                           $ decimal places
Enum65537 =
  0,"DiscreteInput1",
  1,"DiscreteInput2",
  2,"DiscreteInput3",
  3,"DiscreteInput4",
  4,"DiscreteInput5",
  5,"DiscreteInput6",
  6,"DiscreteInput7",
  7,"DiscreteInput8",
  8,"DiscreteInput9",
  9,"DiscreteInput10",
  10,"DiscreteInput11",
  11,"DiscreteInput12",
  12,"DiscreteInput13",
  13,"DiscreteInput14",
  14,"DiscreteInput15",
```

```
    15,"DiscreteInput16";

  Param300005 =
    0,                          $ reserved, shall equal 0
    ,,                          $ Link Path Size, Link Path
    0x0000,                     $ Descriptor
    0xC8,                       $ Data Type
    4,                          $ Data Size in bytes
    "Input Cyclic Interval",    $ name
    "ms",                       $ units
    "",                         $ help string
    10000,10000000,10000,       $ min, max, default data values
    ,,,,                        $ mult, div, base, offset scaling
    ,,,,                        $ mult, div, base, offset links
    ;                           $ decimal places
  Param300006 =
    0,                          $ reserved, shall equal 0
    ,,                          $ Link Path Size, Link Path
    0x0000,                     $ Descriptor
    0xC8,                       $ Data Type
    4,                          $ Data Size in bytes
    "Output Cyclic Interval",   $ name
    "ms",                       $ units
    "",                         $ help string
    10000,10000000,10000,       $ min, max, default data values
    ,,,,                        $ mult, div, base, offset scaling
    ,,,,                        $ mult, div, base, offset links
   ;                            $ decimal places

[Assembly]       $ <TODO: Create assemblies specific to your device here>

  Assem1 = "Output Data",       $ Assembly name
    ,                           $ Path
    4,                          $ Assembly size
    ,,,
    , Param1,                   $ Output Parameter 1
    , Param2;                   $ Output Parameter 2

  Assem65537 = "Input Data",    $ Assembly name
    ,                           $ Path
    2,                          $ Assembly size
    ,,,
    , Param65537;               $ Input Parameter

[Connection Manager]
  $ Connection 1 is an exclusive only connection; Input only not shown in this example
  Connection1 =
     0x04010002,              $ Transport classes = 2
                              $ Production Trigger = Cyclic
                              $ Transport type = Exclusive-owner
                              $ Client
     0x44640405,              $ O->T fixed size supported
                              $ T->O fixed size supported
                              $ O->T Real time transfer format = 32 bit run/idle header
                              $ T->O Real time transfer format = Modeless
                              $ O->T connection type: POINT2POINT
```

```
                              $ T->O connection type: MULTICAST
                              $ T->O connection type: POINT2POINT
                              $ O->T priority: SCHEDULED
                              $ T->O priority: SCHEDULED
    Param300006, , Assem1,    $ O->T RPI, size, format
                              $ <TODO: Insert the O->T RPI, size and format information
                              $ for your connection here>
    Param300005, , Assem65537,    $ T->O RPI, size, format
                              $ <TODO: Insert the T->O RPI, size and format information
                              $ for your connection here>
    ,,                        $ config #1 size, format
    ,,                        $ config #2 size, format
    "<TODO: Insert your connection name here>",      $ Connection Name
    "< TODO: Insert your connection help here >",    $ help string
    "20 04 2E 00 01 00 00 00 2E 00 01 00 01 00";     $ Path
                              $<TODO: Insert the assembly numbers for your
                              $ O->T and T->O data here>
[Modbus Class]
   Revision = 1;
   MaxInst = 1;
   Number_Of_Static_Instances = 1;
   Max_Number_Of_Dynamic_Instances = 0;

   Instance_Services =
         0x4D,                  $ Modbus Function Code 4, Read Input Registers
         0x4E,                  $ Modbus Function Code 3, Read Holding Registers
         0x50,                  $ Modbus Function Code 16 Write Holding Registers
         0x51;                  $ Modbus Passthrough service
```

7-4.4 Text for Example D: Contiguous Block of Configuration Parameters

```
$ Example Description
$ This example is a modification on the Modbus device presented in Example C.  Instead
$ of the configuration parameters being scattered in the Holding Registers table, they
$ are contiguous. Thus, the parameters can be modeled by, and access as, an assembly.
$ This assembly can be written to using explicit messaging, or included in the
$ connection origination process using the Data Segment within a Forward_Open.
$ Application Data:
$ - CfgParam1 - 16-bit unsigned integer
$ - CfgParam2 - 16-bit unsigned integer
$ - CfgParam3 - 16-bit unsigned integer
$ - 16 bits of Discrete Input
$ - 32 bits of Discrete Output
$ Network Presentation:
$ - CfgParam1 is mapped to Modbus holding register 101 and can be written using the
$   Modbus Function Code 16 Write Multiple Registers service and read using the Modbus
$   Function Code 3 Read Holding Registers service.
$ - CfgParam2 is mapped to Modbus holding register 102 and can be written using the
$   Modbus Function Code 16 Write Multiple Registers service and read using the Modbus
$   Function Code 3 Read Holding Registers service.
$ - CfgParam3 is mapped to Modbus holding register 103 and can be written using the
$   Modbus Function Code 16 Write Multiple Registers service and read using the Modbus
$   Function Code 3 Read Holding Registers service.
$ - When accessed, 16 bits of Discrete Input data are packed into Modbus Input Register
$   1, and are read by Modbus Function Code 4, Read Input Registers Service.
$ - When accessed, 32 bits of Discrete Output data are packed into Modbus Holding
$   Registers 1 and 2, and are written by Modbus Function Code 16, Write Multiple
$   Registers Service.

$ EDS Strategy
$ Since the configuration parameters are mapped to a contiguous block of Modbus Holding
$ Registers, they can be mapped to a configuration assembly. The Forward_Open service
$ can also be used to transport the device's configuration assembly in a Data Segment.
$ The [Modbus Class] section acknowledges the device's support for Modbus Function
$ Codes 3, 4, and 16.

[File]
  DescText = "Contiguous Configuration";
  $ <TODO: Update Dates, Times, and Revision>
  CreateDate = ; $ e.g.: 04-04-2007
  CreateTime = ; $ e.g.: 12:00:00
  ModDate = ; $ e.g.: 09-01-2007
  ModTime = ; $ e.g.: 05:05:00
  Revision = 1.2;
  HomeURL = "<TODO: Fill in location of EDS file here, or remove this keyword>";

[Device]
  VendCode = ; $ <TODO: Fill in your VendorID number here> $ Vendor Number
  VendName = "<TODO: Fill in your vendor name here>";      $ Vendor Name
  ProdType = 40;                    $ Device Type code for Modbus devices
  ProdTypeStr = "Modbus Device";    $ Device Type string for Modbus devices
  ProdCode = ; $ <TODO: Fill in this product's code value here>
  MajRev = ; $ <TODO: Fill in this product's major revision value here>
  MinRev = 1;
  ProdName = "<TODO: Fill in your device product name string here>";
```

```
  Catalog = "<TODO: Fill in your catalog string here>";
  Icon = "<TODO: Fill in your icon filename string here>";

  $ Include the ModusIdentityInfo keyword only if your device supports the Read
  $ Device Identification function. If supported, the values entered below shall match
  $ the values returned from that function.
  ModbusIdentityInfo = "TODO: Fill in Vendor name string",
                       "TODO: Fill in Product Code string",
                       "TODO: Fill in Product revision string";


$ The device classification value, ModbusTCP or ModbusSL
[Device Classification]
  Class1 = ; $ <TODO: Fill in ModbusTCP or ModbusSL here>

$ For ParamN Entries, N is defined as follows
$ 0x00000001 - 0x00010000     ( 1 .. 65536)       Holding Registers
$ 0x00010001 - 0x00020000     (65537 .. 131072)   Input Registers
$ 0x00020001 - 0x00030000     (131073 .. 196608)  Coils
$ 0x00030001 - 0x00040000     (196609 .. 262144)  Discrete Inputs

[Params]
  $ Param1 represents the 16 Discrete Outputs at Holding Register 1
  $ Used as Parameter Value attribute, it allows the definition of EDS Enumerations
  $ that map to the first 16 Discrete Outputs of the application data.
  Param1 =
    0,                          $ reserved, shall equal 0
    ,"",                        $ Link Path Size, Link Path
    0x0000,                     $ Descriptor
    0xD2,                       $ Data Type
    2,                          $ Data Size in bytes
    "Discrete Outputs 1 - 16",  $ name
    "",                         $ units
    "16 Discrete Outputs at Holding Register 1",    $ help string
    0x0001,0x8000,0x0000,       $ min, max, default data values
    ,,,,                        $ mult, div, base, offset scaling
    ,,,,                        $ mult, div, base, offset links
    ;                           $ decimal places
  Enum1 =
    0,"DiscreteOutput1",
    1,"DiscreteOutput2",
    2,"DiscreteOutput3",
    3,"DiscreteOutput4",
    4,"DiscreteOutput5",
    5,"DiscreteOutput6",
    6,"DiscreteOutput7",
    7,"DiscreteOutput8",
    8,"DiscreteOutput9",
    9,"DiscreteOutput10",
    10,"DiscreteOutput11",
    11,"DiscreteOutput12",
    12,"DiscreteOutput13",
    13,"DiscreteOutput14",
    14,"DiscreteOutput15",
    15,"DiscreteOutput16";
```

```
$ Param2 represents the 16 Discrete Outputs at Holding Register 2
$ Used as Parameter Value attribute, it allows the definition of EDS Enumerations
$ that map to the next 16 Discrete Outputs of the application data.
Param2 =
  0,                          $ reserved, shall equal 0
  ,"",                        $ Link Path Size, Link Path
  0x0000,                     $ Descriptor
  0xD2,                       $ Data Type
  2,                          $ Data Size in bytes
  "Discrete Outputs 17 - 32", $ name
  "",                         $ units
  "16 Discrete Outputs at Holding Register 2",    $ help string
  0x0001,0x8000,0x0000,       $ min, max, default data values
  ,,,,                        $ mult, div, base, offset scaling
  ,,,,                        $ mult, div, base, offset links
  ;                           $ decimal places
Enum2 =
  0,"DiscreteOutput17",
  1,"DiscreteOutput18",
  2,"DiscreteOutput19",
  3,"DiscreteOutput20",
  4,"DiscreteOutput21",
  5,"DiscreteOutput22",
  6,"DiscreteOutput23",
  7,"DiscreteOutput24",
  8,"DiscreteOutput25",
  9,"DiscreteOutput26",
  10,"DiscreteOutput27",
  11,"DiscreteOutput28",
  12,"DiscreteOutput29",
  13,"DiscreteOutput30",
  14,"DiscreteOutput31",
  15,"DiscreteOutput32";

Param101 =
  0,                        $ reserved, shall equal 0
  , "",                     $ Link Path Size, Link Path
  0x0000,                   $ Descriptor
  0xC7,                     $ Data Type
  2,                        $ Data Size in bytes
  "CfgParam1",              $ name
  "",                       $ units
  "CfgParam1",              $ help string
  ,,0,                      $ min, max, default data values
  ,,,,                      $ mult, div, base, offset scaling
  ,,,,                      $ mult, div, base, offset links
  ;                         $ decimal places
Param102 =
  0,                        $ reserved, shall equal 0
  , "",                     $ Link Path Size, Link Path
  0x0000,                   $ Descriptor
  0xC7,                     $ Data Type
  2,                        $ Data Size in bytes
  "CfgParam2",              $ name
  "",                       $ units
  "CfgParam2",              $ help string
```

```
  ,,0,                        $ min, max, default data values
  ,,,,                        $ mult, div, base, offset scaling
  ,,,,                        $ mult, div, base, offset links
  ;                           $ decimal places
Param103 =
  0,                          $ reserved, shall equal 0
  , "",                       $ Link Path Size, Link Path
  0x0000,                     $ Descriptor
  0xC7,                       $ Data Type
  2,                          $ Data Size in bytes
  "CfgParam3",                $ name
  "",                         $ units
  "CfgParam3",                $ help string
  ,,0,                        $ min, max, default data values
  ,,,,                        $ mult, div, base, offset scaling
  ,,,,                        $ mult, div, base, offset links
  ;                           $ decimal places
$ Param65537 represents the 16 Discrete Inputs at Input Register 1
$ Used as Parameter Value attribute, it allows the definition of EDS Enumerations
$ that map to the 16 Discrete Inputs of the application data.

Param65537 =
  0,                          $ reserved, shall equal 0
  ,"",                        $ Link Path Size, Link Path
  0x0010,                     $ Descriptor = Read Only
  0xD2,                       $ Data Type
  2,                          $ Data Size in bytes
  "Discrete Inputs",          $ name
  "",                         $ units
  "16 Discrete Inputs at Input Register 1",    $ help string
  0x0001,0x8000,0x0000,       $ min, max, default data values
  ,,,,                        $ mult, div, base, offset scaling
  ,,,,                        $ mult, div, base, offset links
  ;                           $ decimal places
Enum65537 =
  0,"DiscreteInput1",
  1,"DiscreteInput2",
  2,"DiscreteInput3",
  3,"DiscreteInput4",
  4,"DiscreteInput5",
  5,"DiscreteInput6",
  6,"DiscreteInput7",
  7,"DiscreteInput8",
  8,"DiscreteInput9",
  9,"DiscreteInput10",
  10,"DiscreteInput11",
  11,"DiscreteInput12",
  12,"DiscreteInput13",
  13,"DiscreteInput14",
  14,"DiscreteInput15",
  15,"DiscreteInput16";

Param300005 =
  0,                          $ reserved, shall equal 0
  ,,                          $ Link Path Size, Link Path
  0x0000,                     $ Descriptor
```

```
    0xC8,                       $ Data Type
    4,                          $ Data Size in bytes
    "Input Cyclic Interval",    $ name
    "ms",                       $ units
    "",                         $ help string
    10000,10000000,10000,       $ min, max, default data values
    ,,,,                        $ mult, div, base, offset scaling
    ,,,,                        $ mult, div, base, offset links
    ;                           $ decimal places
  Param300006 =
    0,                          $ reserved, shall equal 0
    ,,                          $ Link Path Size, Link Path
    0x0000,                     $ Descriptor
    0xC8,                       $ Data Type
    4,                          $ Data Size in bytes
    "Output Cyclic Interval",   $ name
    "ms",                       $ units
    "",                         $ help string
    10000,10000000,10000,       $ min, max, default data values
    ,,,,                        $ mult, div, base, offset scaling
    ,,,,                        $ mult, div, base, offset links
    ;                           $ decimal places

[Assembly]
  Assem1 = "Output Data",       $ Assembly name
    ,                           $ Path
    4,                          $ Assembly size
    ,,,
    , Param1,                   $ Output Parameter 1
    , Param2;                   $ Output Parameter 2

  Assem101 =
    "CfgAssembly",
    ,
    6,
    0x0000,
    ,,
    16,Param101,
    16,Param102,
    16,Param103;

  Assem65537 = "Input Data",    $ Assembly name
    ,                           $ Path
    2,                          $ Assembly size
    ,,,
    , Param65537;               $ Input Parameter

[Connection Manager]
$ Connection 1 is an exclusive only connection; Input only not shown in this example
  Connection1 =
    0x04010002,               $ Transport classes = 2
                              $ Production Trigger = Cyclic
                              $ Transport type = Exclusive-owner
                              $ Client
    0x44640405,               $ O->T fixed size supported
                              $ T->O fixed size supported
```

```
                              $ O->T Real time transfer format = 32 bit run/idle header
                              $ T->O Real time transfer format = Modeless
                              $ O->T connection type: POINT2POINT
                              $ T->O connection type: MULTICAST
                              $ T->O connection type: POINT2POINT
                              $ O->T priority: SCHEDULED
                              $ T->O priority: SCHEDULED
    Param300006, , Assem1,    $ O->T RPI, size, format
                              $ <TODO: Insert the O->T RPI, size and format information
                              $ for your connection here>
    Param300005, , Assem65537,    $ T->O RPI, size, format
                              $ <TODO: Insert the T->O RPI, size and format information
                              $ for your connection here>
    , Assem101,  $ config #1 size, format
    ,,                        $ config #2 size, format
    "<TODO: Insert your connection name here>",    $ Connection Name
    "< TODO: Insert your connection help here >",  $ help string
    "20 04 2E 00 65 00 00 00 2E 00 01 00 00 00 2E 00 01 00 01 00"; $ Path <TODO: Insert
                              $ the assembly numbers for your O->T and T->O data here>

[Modbus Class]
   Revision = 1;
   MaxInst = 1;
   Number_Of_Static_Instances = 1;
   Max_Number_Of_Dynamic_Instances = 0;

   Instance_Services =
          0x4D,                $ Modbus Function Code 4, Read Input Registers
          0x4E,                $ Modbus Function Code 3, Read Holding Registers
          0x50,                $ Modbus Function Code 16 Write Holding Registers
          0x51;                $ Modbus Passthrough service
```

7-4.5 Text for Example E: Modbus/TCP Gateway

```
$ Example Description
$ A Modbus/TCP Gateway is a device with at least one Modbus/TCP port and one Modbus
$ Serial port.  This device can route messages from Modbus/TCP to Modbus Serial.  The
$ gateway may also have valid data items for reading/writing (including input and
$ output data). In this example, there is one configuration register and one status
$ register available.
$ Application Data:
$ - CfgParam - 16-bit unsigned integer
$ - StatusParam - 16-bit unsigned integer
$ Network Presentation:
$ - CfgParam is mapped to Modbus holding register 1 and can be written using the Modbus
$   Function Code 16 Write Multiple Registers service and read using the Modbus
$   Function Code 3 Read Holding Registers service.
$ - StatusParam is mapped to Modbus holding register 2 and can be read using the Modbus
$   Function Code 3 Read Holding Registers service.
$ EDS Strategy
$ The Port section of the EDS is used to indicate the presence of additional Modbus
$ ports to the configuration tool.  In this example, the device declares two ports in
$ the Port section, one for Modbus/TCP and one for Modbus Serial.  The EDS also defines
$ the configuration and status parameters available.
$ The [Modbus Class] section acknowledges the device's support for Modbus Function
$ Codes 3 and 16.

[File]
  DescText = "Modbus/TCP Gateway";
  $ <TODO: Update Dates, Times, and Revision>
  CreateDate = ; $ e.g.: 09-07-2007
  CreateTime = ; $ e.g.: 12:00:00
  ModDate = ; $ e.g.: 09-07-2007
  ModTime = ; $ e.g.: 09:07:00
  Revision = 1.1;
  HomeURL = "<TODO: Fill in location of EDS file here, or remove this keyword>";

[Device]
  VendCode = ; $ <TODO: Fill in your VendorID number here> $ Vendor Number
  VendName = "<TODO: Fill in your vendor name here>";      $ Vendor Name
  ProdType = 40;                   $ Device Type code for Modbus devices
  ProdTypeStr = "Modbus Device";   $ Device Type string for Modbus devices
  ProdCode = ; $ <TODO: Fill in this product's code value here>
  MajRev = ; $ <TODO: Fill in this product's major revision value here>
  MinRev = 1;
  ProdName = "<TODO: Fill in your device product name string here>";
  Catalog = "<TODO: Fill in your catalog string here>";
  Icon = "<TODO: Fill in your icon filename string here>";

  $ Include the ModusIdentityInfo keyword only if your device supports the Read
  $ Device Identification function. If supported, the values entered below shall match
  $ the values returned from that function.
  ModbusIdentityInfo = "TODO: Fill in Vendor name string",
                       "TODO: Fill in Product Code string",
                       "TODO: Fill in Product revision string";
```

```
[Device Classification]
   Class1 = ModbusTCP;        $ Code is for Modbus TCP Devices
   Class2 = ModbusSL;         $ Code is for Modbus Serial Devices

[Port]
   Port1 = ModbusTCP,
           "MBTCP Port",,2;
   Port2 = ModbusSL,
           "MBSL Port",,1;

$ For ParamN Entries, N is defined as follows
$ 0x00000001 - 0x00010000     ( 1 .. 65536)       Holding Registers
$ 0x00010001 - 0x00020000     (65537 .. 131072)   Input Registers
$ 0x00020001 - 0x00030000     (131073 .. 196608)  Coils
$ 0x00030001 - 0x00040000     (196609 .. 262144)  Discrete Inputs

[Params]
   Param1 =
     0,                       $ reserved, shall equal 0
     ,"",                     $ Link Path Size, Link Path
     0x0000,                  $ Descriptor
     0xC7,                    $ Data Type
     2,                       $ Data Size in bytes
     "CfgParam",              $ name
     "",                      $ units
     "CfgParam",              $ help string
     ,,0,                     $ min, max, default data values
     ,,,,                     $ mult, div, base, offset scaling
     ,,,,                     $ mult, div, base, offset links
     ;                        $ decimal places
   Param2 =
     0,                       $ reserved, shall equal 0
     ,"",                     $ Link Path Size, Link Path
     0x0010,                  $ Descriptor
     0xC7,                    $ Data Type
     2,                       $ Data Size in bytes
     "StatusParam",           $ name
     "",                      $ units
     "StatusParam",           $ help string
     ,,0,                     $ min, max, default data values
     ,,,,                     $ mult, div, base, offset scaling
     ,,,,                     $ mult, div, base, offset links
     ;                        $ decimal places

[Modbus Class]
        Revision = 1;
        MaxInst = 1;
        Number_Of_Static_Instances = 1;
        Max_Number_Of_Dynamic_Instances = 0;
        Instance_Services =
                0x4E,         $ Modbus Function Code 3, Read Holding Registers
                0x50,         $ Modbus Function Code 16, Write Holding Registers
                0x51;         $ Modbus Passthrough service
```

7-5 Text for Generic Modbus Device EDS File

```
[File]
  DescText = "Generic EDS to support Modbus Devices without an EDS";
  CreateDate = 04-04-2007;
  CreateTime = 12:00:00;
  ModDate = 09-01-2007;
  ModTime = 04:50:00;
  Revision = 1.2;
  HomeURL = "<TODO: Fill in location of EDS file here, or remove this keyword>";

$ This EDS is for General Use with Modbus Devices
[Device]
  VendCode = 65534;               $ This is the Default Modbus Device Vendor_ID.
  VendName = "Modbus Device Vendor";
  ProdType = 40;                  $ Device Type code for Modbus devices
  ProdTypeStr = "Modbus Device";  $ Device Type string for Modbus devices
  ProdCode = 1;
  MajRev = 1;
  MinRev = 1;
  ProdName = "Generic Modbus Device";
  Icon = "<TODO: Fill in your icon filename string here>";

$ The device classification value, ModbusTCP and ModbusSL
[Device Classification]
  Class1 = ModbusTCP;              $ Code is for Modbus TCP Devices
  Class2 = ModbusSL;               $ Code is for Modbus Serial Devices

$ For ParamN Entries, N is defined as follows
$ 0x00000001 - 0x00010000     ( 1 .. 65536)       Holding Registers
$ 0x00010001 - 0x00020000     (65537 .. 131072)   Input Registers
$ 0x00020001 - 0x00030000     (131073 .. 196608)  Coils
$ 0x00030001 - 0x00040000     (196609 .. 262144)  Discrete Inputs
$ (300000 - ?) Internal EDS parameters
[Params]
  Param300001 =
    0,                            $ reserved, shall equal 0
    ,,                            $ Link Path Size, Link Path
    0x0004,                       $ Descriptor
    0xC7,                         $ Data Type
    2,                            $ Data Size in bytes
    "Number of Input Registers",  $ name
    "",                           $ units
    "",                           $ help string
    0,250,16,                     $ min, max, default data values
    1,2,1,0,                      $ mult, div, base, offset scaling
    ,,,,                          $ mult, div, base, offset links
    ;                             $ decimal places

  Param300002 =
    0,                            $ reserved, shall equal 0
    ,,                            $ Link Path Size, Link Path
    0x0004,                       $ Descriptor
    0xC7,                         $ Data Type
```

```
  2,                              $ Data Size in bytes
  "Number of Output Registers",          $ name
  "WORD",                                $ units
  "Number of 16 bit output registers",   $ help string
  0,246,16,                       $ min, max, default data values
  1,2,1,0,                        $ mult, div, base, offset scaling
   ,,,,                           $ mult, div, base, offset links
  ;                               $ decimal places

Param300003 =
  0,                              $ reserved, shall equal 0
  ,,                              $ Link Path Size, Link Path
  0x0004,                         $ Descriptor
  0xC8,                           $ Data Type
  4,                              $ Data Size in bytes
  "Starting Input Element Number in Input Register Table",   $ name
  "",                                                        $ units
  "Mapped as Input Register in the Input Register Table",    $ help string
  0x00010001,0x00020000,0x00010001,    $ min, max, default data values
  1,1,1,0x10000,                  $ mult, div, base, offset scaling
  ,,,,                            $ mult, div, base, offset links
  ;                               $ decimal places

Param300004 =
  0,                              $ reserved, shall equal 0
  ,,                              $ Link Path Size, Link Path
  0x0000,                         $ Descriptor
  0xC8,                           $ Data Type
  4,                              $ Data Size in bytes
  "Starting Output Element Number in Output Register Table",    $ name
  "",                             $ units
  "Mapped as Output Register in the Holding Register Table",    $ help string
  0x00000001,0x00010000,0x00000001,    $ min, max, default data values
  ,,,,                            $ mult, div, base, offset scaling
  ,,,,                            $ mult, div, base, offset links
  ;                               $ decimal places

Param300005 =
  0,                              $ reserved, shall equal 0
  ,,                              $ Link Path Size, Link Path
  0x0000,                         $ Descriptor
  0xC8,                           $ Data Type
  4,                              $ Data Size in bytes
  "Input Cyclic Interval",        $ name
  "ms",                           $ units
  "",                             $ help string
  10000,10000000,10000,           $ min, max, default data values
  ,,,,                            $ mult, div, base, offset scaling
  ,,,,                            $ mult, div, base, offset links
  ;                               $ decimal places

Param300006 =
  0,                              $ reserved, shall equal 0
  ,,                              $ Link Path Size, Link Path
  0x0000,                         $ Descriptor
  0xC8,                           $ Data Type
```

```
  4,                              $ Data Size in bytes
  "Output Cyclic Interval",       $ name
  "",                             $ units
  "",                             $ help string
  10000,10000000,10000,           $ min, max, default data values
  ,,,,                            $ mult, div, base, offset scaling
  ,,,,                            $ mult, div, base, offset links
  ;                               $ decimal places

Param300007 =
  0,                              $ reserved, shall equal 0
  ,,                              $ Link Path Size, Link Path
  0x0000,                         $ Descriptor
  0xC8,                           $ Data Type
  4,                              $ Data Size in bytes
  "Starting Input Element Number in Holding Register Table",   $ name
  "",                             $ units
  "Mapped as Input Register in the Holding Register Table",    $ help string
  0x00000001,0x00010000,0x00000001,    $ min, max, default data values
  ,,,,                            $ mult, div, base, offset scaling
  ,,,,                            $ mult, div, base, offset links
  ;                               $ decimal places

Param300008 =
  0,                              $ reserved, shall equal 0
  ,,                              $ Link Path Size, Link Path
  0x0004,                         $ Descriptor
  0xC7,                           $ Data Type
  2,                              $ Data Size in bytes
  "Number of Contiguous Configuration Registers",          $ name
  "WORD",                         $ units
  "Number of 16 bit contiguous configuration registers",   $ help string
  0, 246, 0,                      $ min, max, default data values
  1,2,1,0,                        $ mult, div, base, offset scaling
  ,,,,                            $ mult, div, base, offset links
  ;                               $ decimal places

Param300009 =
  0,                              $ reserved, shall equal 0
  ,,                              $ Link Path Size, Link Path
  0x0000,                         $ Descriptor
  0xC8,                           $ Data Type
  4,                              $ Data Size in bytes
  "Starting Configuration Element Number in Holding Register Table",  $ name
  "",                             $ units
  "Mapped as Data in the Holding Register Table",                     $ help string
  0x00000001,0x00010000,0x00000001,    $ min, max, default data values
  ,,,,                            $ mult, div, base, offset scaling
  ,,,,                            $ mult, div, base, offset links
  ;                               $ decimal places
```

```
[Connection Manager]
  $ For Connection 1
  $ O->T is a connection to the specified Holding Register(s)
  $ T->O is a connection to the specified Input Register(s)
  Connection1 =
    0x04010002,   $ Transport classes = 2
                  $ Production Trigger = Cyclic
                  $ Transport type = Exclusive-owner
                  $ Client

    0x44640405,   $ O->T fixed size supported
                  $ T->O fixed size supported
                  $ O->T Real time transfer format = 32 bit run/idle header
                  $ T->O Real time transfer format = Modeless
                  $ O->T connection type: POINT2POINT
                  $ T->O connection type: MULTICAST
                  $ T->O connection type: POINT2POINT
                  $ O->T priority: SCHEDULED
                  $ T->O priority: SCHEDULED
    Param300006,Param300002,,    $ O->T RPI, size, format
    Param300005,Param300001,,    $ T->O RPI, size, format
    Param300008,,                $ config #1 size, format
    ,,                           $ config #2 size, format
    "I/O connection with inputs from input registers",        $ Connection Name
    "Parameterized Implicit Connection to Assembly Objects",  $ help string
    "20 04 26 00 [Param300009] 2E 00 [Param300004] 2E 00 [Param300003]"; $ Path

  $ For Connection 2
  $ O->T is a heartbeat
  $ T->O connection is to the specifi of Input Registers.
  Connection2 =
    0x02010002,   $ Transport classes = 2
                  $ Production Trigger = Cyclic
                  $ Transport type = Input-only
                  $ Client
    0x44640405,   $ O->T fixed size supported
                  $ T->O fixed size supported
                  $ O->T Real time transfer format = 32 bit run/idle header
                  $ T->O Real time transfer format = Modeless
                  $ O->T connection type: POINT2POINT
                  $ T->O connection type: MULTICAST
                  $ T->O connection type: POINT2POINT
                  $ O->T priority: SCHEDULED
                  $ T->O priority: SCHEDULED
     ,0,,                        $ O->T RPI, size, format
     Param300005,Param300001,,   $ T->O RPI, size, format
     ,,                          $ config #1 size, format
     ,,                          $ config #2 size, format
     "Input only connection with inputs from input registers",   $ Connection Name
     "Parameterized Implicit Connection to Assembly Objects",    $ help string
     "20 04 26 00 [Param300009] 2E 00 00 00 00 00 2E 00 [Param300003]";    $ Path
```

```
$ For Connection 3
$ O->T is a connection to the 1st Holding Register of and
$ T->O connection is to the 1st register of Holding Registers.
Connection3 =
  0x04010002,   $ Transport classes = 2
                $ Production Trigger = Cyclic
                $ Transport type = Exclusive-owner
                $ Client
  0x44640405,   $ O->T fixed size supported
                $ T->O fixed size supported
                $ O->T Real time transfer format = 32 bit run/idle header
                $ T->O Real time transfer format = Modeless
                $ O->T connection type: POINT2POINT
                $ T->O connection type: MULTICAST
                $ T->O connection type: POINT2POINT
                $ O->T priority: SCHEDULED
                $ T->O priority: SCHEDULED
  Param300006,Param300002,,    $ O->T RPI, size, format
  Param300005,Param300001,,    $ T->O RPI, size, format
  ,,                           $ config #1 size, format
  ,,                           $ config #2 size, format
  "I/O connection with inputs from holding registers",      $ Connection Name
  "Parameterized Implicit Connection to Assembly Objects",  $ help string
  "20 04 26 00 [Param300009] 2E 00 [Param300004] 2E 00 [Param300007]";  $ Path

$ For Connection 4
  $ O->T is a heartbeat and
  $ T->O connection is to the 1st register of Holding Registers.
      Connection4 =
        0x02010002,   $ Transport classes = 2
                      $ Production Trigger = Cyclic
                      $ Transport type = Input-only
                      $ Client

        0x44640405,   $ O->T fixed size supported
                      $ T->O fixed size supported
                      $ O->T Real time transfer format = 32 bit run/idle header
                      $ T->O Real time transfer format = Modeless
                      $ O->T connection type: POINT2POINT
                      $ T->O connection type: MULTICAST
                      $ T->O connection type: POINT2POINT
                      $ O->T priority: SCHEDULED
                      $ T->O priority: SCHEDULED
        ,0,,                          $ O->T RPI, size, format
        Param300005,Param300001,,     $ T->O RPI, size, format
        ,,                            $ config #1 size, format
        ,,                            $ config #2 size, format
        "Input only connection with inputs from holding registers", $ Connection Name
        "Parameterized Implicit Connection to Assembly Objects",    $ help string
        "20 04 26 00 [Param300009] 2E 00 00 00 00 00 2E 00 [Param300007]";    $ Path
```

```
[Modbus Class]
   Revision = 1;
   MaxInst = 1;
   Number_Of_Static_Instances = 1;
   Max_Number_Of_Dynamic_Instances = 0;

   Instance_Services =
      0x4B,                    $ Modbus Function Code 2, Read Discrete Inputs
      0x4C,                    $ Modbus Function Code 1, Read Coils
      0x4D,                    $ Modbus Function Code 4, Read Input Registers
      0x4E,                    $ Modbus Function Code 3, Read Holding Registers
      0x4F,                    $ Modbus Function Code 15, Write Multiple Coils
      0x50,                    $ Modbus Function Code 16, Write Holding Registers
      0x51;                    $ Modbus Passthrough
```

This page is intentionally left blank

Volume 7: Integration of Modbus Devices into the CIP Architecture

Chapter 8: Physical Layer

Contents

8-1 Introduction

This chapter is for additions to the physical layer that are specific to Modbus device integration. At this time, no such additions exist.

This page is intentionally left blank

Volume 7: Integration of Modbus Devices into the CIP Architecture

Chapter 9: Indicators and Middle Layers

Contents

9-1 Introduction

This chapter is for additions to the indicators and middle layers that are specific to Modbus device integration. At this time, no such additions exist.

This page is intentionally left blank

Volume 7: Integration of Modbus Devices into the CIP Architecture

Chapter 10: CIP Bridging and Routing

Contents

10-1 Introduction

This chapter identifies the behavior associated with translating CIP communications with Modbus. This behavior, which is exhibited within a component called the translation function, is independent of where the function is located. The translation function may exist in a separate routing (linking) device, or may be integrated into a CIP originator. The device containing the translation function is called a translator

The communication objects within Volume 1 Chapter 3, in particular services of the Connection Manager, provide routing between CIP subnets. This chapter will describe how CIP to Modbus translation is accomplished using those objects when communicating with a target device on Modbus.

10-2 CIP to Modbus Data Translation

CIP to Modbus translators handle the conversion of CIP messaging to Modbus functions. For additional information on data translation between CIP and Modbus, see Chapter 6 Device Profiles within this volume.

10-2.1 Modbus Data Model Representation in CIP

10-2.1.1 Assembly and Parameter Object Representation of Bit Addressable Tables

The bit addressable Modbus tables (Discrete Input and Coils) are represented in the assembly object as an array of bits. Each member in the assembly is one bit in length, representing one data item in the Modbus table. The assembly object instances are implemented such that each instance represents a bit offset into the table based on the instance number (one based and relative to the starting instance offset depending on the Modbus table represented; see Chapters 5 and 6). The number of members in each Assembly instance is either 2000, or the number of members remaining to the end of the Modbus table if less than 2000 data items remain. This results in a maximum assembly size of 250 bytes for each instance.

The Parameter Object instances use the same numbering mechanism as the Assembly Object, and represent a single data item (one bit in the table as indicated by the Parameter Object instance number).

10-2.1.2 Assembly and Parameter Object Representation of Word Addressable Tables

The word addressable Modbus tables (Input Registers and Holding Registers) are represented in the assembly object as an array of 16 bit words. Each member in the assembly is 16 bits in length, representing one data item in the Modbus table. The assembly object instances are implemented such that each instance represents a word offset into the table based on the instance number (one based and relative to the starting instance offset depending on the Modbus table represented). The number of members in each Assembly instance is either 125, or the number of members remaining to the end of the table if less than 125 data items remain. This results in a maximum assembly size of 250 bytes for each instance.

The Parameter Object instances use the same numbering mechanism as the Assembly Object, and represent a single data item (one word in the table as indicated by the Parameter Object instance number).

10-2.2 Translation of Identity Information

See Chapter 5 (the Identity Object) for information on how the translation function translates requests for identity information.

10-3 Translation of CIP Services to Modbus Functions

I/O and Explicit Messaging services are translated to Modbus based on the CIP EPATH and CIP Service specified. These parameters indicate which Modbus table to interface with, and the direction of the request.

The tables below show the relationship between CIP and Modbus functions. Only those CIP Services and EPATH combinations indicated in the table shall be routed to Modbus. Table 10-3.1 shows the CIP to Modbus translation for explicit messaging requests (both unconnected and across an Explicit Messaging connection). Table 10-3.2 shows the application objects which can be connected to using the Forward_Open service (and resulting in either an Implicit or Explicit connection), and the CIP to Modbus translation for the resulting connections.

Table 10-3.1 CIP to Modbus Translation of Explicit Messaging Requests

EPATH Class	EPATH Instance	CIP Service	Modbus Function
Assembly Object	0x00000001 – 0x00010000	Get_Member	Read Holding Registers (0x03)
		Set_Member	Write Single Register (0x06) Write Multiple Registers (0x10)
	0x00010001 – 0x00020000	Get_Member	Read Input Registers (0x04)
	0x00020001 – 0x00030000	Get_Member	Read Coils (0x01)
		Set_Member	Write Single Coil (0x05) Write Multiple Coils (0x0F)
	0x00030001 – 0x00040000	Get_Member	Read Discrete Inputs (0x02)
Parameter Object	0x00000001 – 0x00010000	Get_Attribute_Single	Read Holding Registers (0x03)
		Set_Attribute_Single	Write Single Register (0x06) Write Multiple Registers (0x10)
	0x00010001 – 0x00020000	Get_Attribute_Single	Read Input Registers (0x04)
	0x00020001 – 0x00030000	Get_Attribute_Single	Read Coils (0x01)
		Set_Attribute_Single	Write Single Coil (0x05) Write Multiple Coils (0x0F)
	0x00030001 – 0x00040000	Get_Attribute_Single	Read Discrete Inputs (0x02)
Identity Object	1	Get_Attribute_Single Get_Attribute_All	Read Device Identification (0x2B/0x0E)

Table 10-3.2 Target Application Objects and the Resulting CIP to Modbus Translation

EPATH Class	EPATH Instance	Messaging Type	Modbus Function
Assembly Object	0x00000001 – 0x00010000	I/O Consumption[1]	Read Holding Registers (0x03)
		I/O Production[2]	Write Single Register (0x06) Write Multiple Registers (0x10)
		I/O Produce/Consume[3]	Read/Write Multiple Registers[4] (0x17)
	0x00010001 – 0x00020000	I/O Consumption[1]	Read Input Registers (0x04)
Message Router Object	1	Explicit Messaging Produce/Consume[5]	Modbus function depends on the explicit request received across the connection. See Table 10-3.1

1 Using Transport Class 0/1, Target to Originator direction
2 Using Transport Class 0/1, Originator to Target direction
3 Using Transport Class 3
4 This function shall only be used when the O_to_T and T_to_O RPI values are the same
5 Using Transport Class 3, Originator is client

10-4 Translated Messages to Modbus

Modbus data can be accessed using connected messaging to accomplish I/O transactions and both connected and unconnected messaging for Explicit Message transactions.

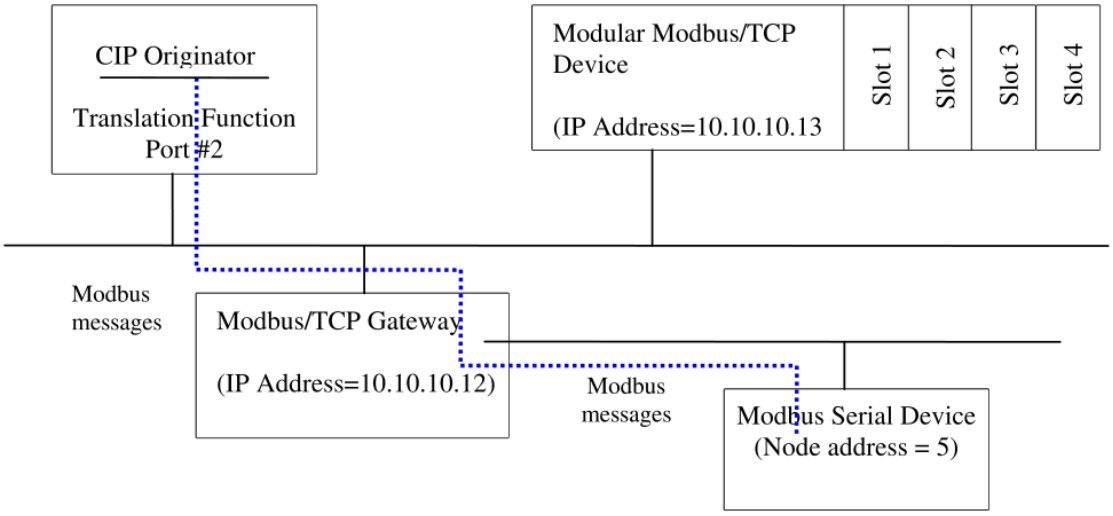
10-4.1 Translation Rules

10-4.1.1 Use of Unit Identifier (Modbus/TCP)

When the translator is present on Modbus/TCP an additional network hop may be present in the connection path (EPATH) and shall be supported by the translation function. This additional hop is specified using Port #1 (the backplane port) as the port number and a one byte node address. This node address is placed into the Modbus/TCP Unit Identifier field in the MBAP Header. If no additional hop is specified, then the translation function shall place the value 0xFF into the Unit Identifier field.

The following diagram shows a network where the Unit ID field is used to address both a Modbus Serial device connected to a Modbus/TCP gateway and a module (slot) within a modular Modbus/TCP device.

Figure 10-4.1 Unit Identifier Example



The connection path seen by the translation function for a message destined to the Modbus Serial device is: [2, “10.10.10.12”, 1, 5]; where “2” is the port number for the Modbus/TCP port on the originator, “10.10.10.12” is the IP address of the Modbus/TCP Gateway, “1” is the backplane port for the gateway, and “5” is the node address of the Modbus Serial device that is the target of the message.

The connection path seen by the translation function for a message destined to a module within Slot 1 of the Modular Modbus/TCP device is: [2, “10.10.10.13”, 1, 1]; where “2” is the port number for the Modbus/TCP port on the originator, “10.10.10.13” is the IP address of the Modular Modbus/TCP device, “1” is the backplane port for the modular device, and “1” is the slot number of the module that is the target of the message.

10-4.1.2 Modbus Port Addressing

A Modbus/TCP or Modbus Serial port is represented by an instantiated Port Object instance, and has a unique Port Number. The Port Type within the Port Object instance indicates either Modbus/TCP or Modbus Serial. Routing through the translator to a Modbus target device makes use of the Port Number assigned by the device containing the translation function.

See the Port Object in Volume 1, Chapter 3 for more information on Port Types, Port Numbers, and the Port Object.

10-4.1.3 Use of Multiple Write Function

When issuing a write only Modbus function, the translation function shall always attempt the write multiple function code (Write Multiple Registers or Write Multiple Coils) first. This will occur when writing to the output tables (Holding Registers or Coils). If the write multiple fails with a Modbus Illegal Function exception code, and the value to write is one data unit in length, the translation function shall attempt to use the write single function code (Write Single Register or Write Single Coil).

10-4.1.4 Translation of Forward_Open Service Parameters

The translation function shall translate the Forward_Open service parameters are described below. Any field values received that are not specified below shall result in an error response.

- Network Connection Parameter (Connection Size): Specified in bytes, this parameter indicates either the exact or the maximum amount of data that will be sent in each CIP message (see Fixed/Variable Network Connection Parameter below). For I/O connections, the translation function shall translate this to the correct number of data items (16 bit registers) used in the Modbus function request.
- Network Connection Parameter (Fixed/Variable): I/O connections shall be set to fixed, Explicit Messaging connections shall be set to variable. Otherwise, an error is returned.
- Network Connection Parameter (Priority): The translation function shall support all priority values that are valid for the associated CIP network. The implementation of these settings on the Modbus network is product specific.
- Network Connection Parameter (Connection Type): The translation function shall support Point to Point and Multicast (in the T->O direction).

  For a Multicast connection request, the translation function checks the connection request to see if it matches an existing connection. If a match is found, the translation function returns the same T->O Network Connection ID as the existing connection (for EthernNet/IP, it also returns the IP multicast address). When the T->O production trigger occurs, the translation function does the appropriate Modbus read to get the data, then sends the T->O data multicast (as defined by the CIP network) to the connection originator(s).

  A connection request matches an existing connection when the encoded application path (in the Forward_Open Connection_Path) points to the same starting Modbus data item and both the connection size (in the Forward_Open Network Connection Parameters) and RPI match exactly.
- Network Connection Parameter (Redundant Owner): This value shall be 0, otherwise an error is returned.
- Transport Type/Trigger: Transport types 0, 1, and 3 shall be supported. When the client indication is set to true, only a trigger of Cyclic shall be supported.

10-4.1.5 Restrictions on I/O and Configuration Application Paths

The translation function (and thus the Modbus target device) shall only support Output (O->T) and Configuration application connection paths to the following:

- Assembly Objects for the Holding Registers (Instances 0x0001 – 0x10000)

The translation function (and thus the Modbus target device) shall only support Input (T->O) application connection paths to the following:

- Assembly Objects for the Holding Registers (Instances 0x0001 – 0x10000)
- Assembly Objects for the Input Registers (Instances 0x10001 – 0x20000)

10-4.1.6 Support of Logical EPATH Sizes

The translation function shall support all valid Logical Format values when parsing an EPATH field.

10-4.1.7 Electronic Key Checking

The translation function shall accept the following electronic key values as valid:

- No electronic key (Electronic Key Segment not present)
- Null Electronic Key (all key values are zero)
- Electronic Key for Device Type only (all other key values are zero), when the Device Type is Modbus Device (Device Type = 0x28)

10-4.1.8 Support of Data Segment

If a connection request contains a Data Segment and a corresponding configuration application connection path, the translation function shall send the data within the Data Segment to the data item(s) specified in the connection path, using the appropriate Modbus write function. This data shall be sent before verifying the T->O connection size parameter with the Modbus target device.

10-4.1.9 Run/Idle Support and the Real Time Format

All Modbus target devices are considered to have the following Real Time formats:

- O->T: 32-bit header format, includes run/idle notification
- T->O: Modeless format, no run/idle notification

In the O->T direction, the translation function shall remove the 32-bit header and evaluate it. If an IDLE condition is indicated, the translation function shall not send the remaining O->T data to the Modbus target device. T->O data transmission shall continue.

If the translation function is using the Modbus Read/Write Multiple Registers function (Function Code 0x23) when communicating with the target device during a Run indication, a switch to the Read Multiple Register function will need to occur during an Idle indication to continue reading the input data. A Read/Write function with zero length output data is not allowed.

10-4.1.10 Heartbeat Messages

If a connection request specifies zero length application data for either direction (a heartbeat connection), the translation function shall ignore the applicable connection path. When a heartbeat connection exists as on of the connection pair, the translation function shall not use the Read/Write function code (Function Code 0x23). The following additional behavior is provided by the translation function based on the direction of the zero length data connection:

- T->O Direction: The translation function shall not issue any Read requests to the target module. At each RPI time, a zero length message is returned to the originator.
- O->T Direction: The translation function ignores the 32-bit header information, and shall not issue any Write requests to the target device.

10-4.1.11 Endian Issues

The translation function shall convert word (16 bit) fields between big endian (on Modbus) and little endian (on CIP). This includes both “2 byte” function parameters and 16 bit data items (Input and Holding register data).

10-4.1.12 Error Management Principles for Modbus/TCP

For connected messaging the inactivity between the originator and the translation function is managed using the CIP Inactivity/Watchdog Timer. The inactivity timeout between the translation function and the Modbus target device is managed by the standard TCP mechanisms. In addition some Modbus Application Layer Timeout mechanism may be implemented in order to have a Modbus Application Layer fault detection that can be quicker than the TCP level fault detection. If this mechanism is implemented it shall be in conformance with the description provided in the Functional Description chapter (Chapter 4) of the "Modbus Messaging on TCP/IP Implementation Guide".

Detailed description of Connected I/O Messaging error management is provided in Chapters 10-4.2.9 and 10-4.2.10

Detailed description of Unconnected Explicit Messaging error management is provided in Chapters 10-4.4.4.1 and 10-4.4.5.1.

Detailed description of Connected Explicit Messaging error management is provided in Chapters 10-4.4.5.3 and 10-4.4.5.4.

10-4.1.13 Linkage with TCP Connections for Modbus/TCP

Modbus/TCP Messages are transmitted over a TCP connection to the Modbus target device. The translation function shall open at least one TCP connection to each Modbus target device. The translation function shall open the TCP connection on the reception of either a Forward_Open or an UnconnectedSend service to the Connection Manager object, if no TCP connection is already established with the Modbus target device.

If the TCP connection is closed by the local or remote TCP/IP stack for any reason the translation function shall delete the associated CIP connections.

NOTES:

For Connected (I/O and Explicit) Messaging the translation function may open a new TCP connection at each reception of a Forward_Open request with the same Modbus target device.

The translation function shall attempt to open separate TCP connections for Explicit Messages and I/O Messages sent to the same Modbus target device (one for Explicit Messages and one for I/O Messages) in order to avoid the risk of starvation of one data flow by the other. If the target device does not support opening of more than one TCP connection with the translation function, then the translation function shall use that single connection for all messaging.

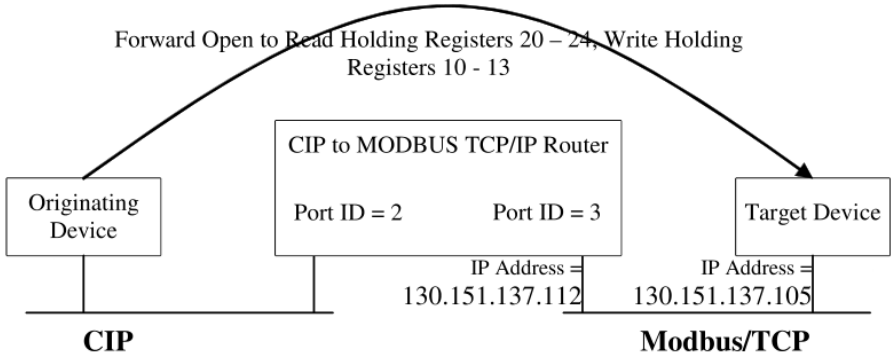
This standard defines no requirements for the management of the TCP connection, such as inactivity timeouts or closing of the TCP connection when all CIP connections are closed. However, implementations are free to implement these.

10-4.2 Connected I/O Messaging – Single Hop

Connected I/O Messaging is established using the ForwardOpen service of the Connection Manager object. The CIP / Modbus translation function acts as the proxy for the Modbus target device by processing and responding to the connection request, and translating the subsequent CIP connection data to Modbus function requests.

Assume that the *Originating Device* wants to make an I/O connection to Read Holding Registers 20 – 24, and write Holding Registers 10 - 13.

Figure 10-4.2 Forward_Open to Establish I/O Connection



10-4.2.1 Step 1 – Request delivered to the CIP to Modbus TCP/IP Translation function

The first step in the process calls for the Originating Device to issue a Forward_Open request. This example uses two transport class 1 connections; one for I/O production and one for I/O consumption. The table below presents the contents of the Service Data field for the Forward_Open request which is used to execute the connection establishment example noted above (all values are in hex).

Table 10-4.1 Forward_Open Request Service Data

Byte Value	Meaning
pp	Priority/Tick Time field.
tt	Connection Timeout Ticks field.
FF FF FF FF	O_to_T Connection ID– In this example, chosen by target node[2]
00 00 00 00	T_to_O Connection ID – In this example, chosen by originating node[2]
0C 00	Connection Serial Number
FF FF	Originator Vendor ID (Vendor ID = 65535)
00 01 02 03	Originator Serial Number (Serial Number = 0x03020100)
00	Connection Timeout Multiplier
00 00 00	Reserved
20 A1 07 00	O_to_T RPI (0x7A120 = 500 ms)
44 0C	O_to_T Connection Parameters[1] This parameter value indicates that the consumed connection size is 12 bytes[3], is fixed size, High Priority, and Point to Point.
20 A1 07 00	T_to_O RPI (0x7A120 = 500 ms)
44 0A	T_to_O Connection Parameters[1] This parameter value indicates that the produced connection size is 10 bytes, is fixed size, High Priority, and Point to Point.

Byte Value	Meaning
01	Transport Type / Trigger (Class 1 Cyclic Client)
11	Connection_Path_Size field indicates that there are 17 words in the connection path.
13 0F 31 33 30 2E 31 35 31 2E 31 33 37 2E 31 30 35 00 21 00 04 00 25 00 14 00 21 00 04 00 25 00 0A 00	Contents of the Connection_Path field. This field specifies the routing/connection information. These bytes indicate that the request is to be delivered out Port #3 and to IP Address 130.151.137.105. This path encoding uses the Extended Link Address format for the IP address. It further indicates the applications being connected to by specifying 16 bit Class ID = 4, 16 bit Instance ID = 10 for consumption (O to T), and 16 bit Class ID = 4, 16 bit Instance ID = 20 for production (T to O). Note that for the Assembly Object, the attribute is assumed to be the Data attribute (Attribute ID 3) if not present in the EPATH.

1 When connecting to a Word Addressable table the connection size parameter must be a multiple of 2. If not, a General Status code of 0x01 and an Additional Status code of 0x0109 (Invalid Connection Size) shall be returned.
2 The allocation of the Connection ID between target and originator is determined by the particular CIP network requirements.
3 The connection size includes 4 bytes for the Real Time Format header.

The Forward_Open request has now been delivered to the CIP/Modbus translation function.

10-4.2.2 Step 2 – Translation function creates resources for connection with the Originator

Upon receipt of a Forward_Open to the Connection Manager object, the translation function shall create the CIP connection resources to be able to establish and maintain a connection between itself and the originating device. If the resources are unavailable, an error response is returned.

10-4.2.3 Step 3 – Translation function creates resources for connection with the Target

After the Connection Manager Object within the CIP translation function processes the Forward_Open request it examines the contents of the Connection_Path field. In the example above, the data in this field indicates that the next device in the hop is on Port #3 and its Node (IP) Address is 130.151.137.105 on Modbus/TCP. Additionally, since there is only a single Port/Node Address pair specified in the Connection_Path field, the request has reached its destination network. When a CIP translation function receives a Forward_Open Request that does not need to be forwarded through more intermediate CIP Routers, the following steps are taken:

- The translation function executes the timing related logic associated with the Priority/Tick Time and Connection Timeout Ticks fields. If a timeout is detected, then an unsuccessful Forward_Open response that specifies a Routing Error is returned to the Requesting Device.
- The translation function verifies that the target application paths are valid for a Modbus target device.
- The translation function creates the connection resources for maintaining communication with the Modbus Target device. If the resources are unavailable, an error response is returned to the originator.

10-4.2.4 Step 4 – Process Configuration Data, if Data Segment is Present

If a Data Segment is present, send the data within the Data Segment to the data item(s) specified in the connection path to the target Modbus device. If an error is returned from the target device, the translation function shall return a Forward_Open error response. The error response shall be General Status 0x01, Additional Status 0x0118 (Invalid Configuration Format). Possible errors are:

- Configuration path specifies a data area that is not writeable
- Target device does not support write requests
- Data area written to invalid

10-4.2.5 Step 5 – Validate Electronic Key

If an Electronic Key Segment is present, the translation function shall verify that the electronic key is valid. See section 10-4.1.7 for valid electronic key values.

10-4.2.6 Step 6 – Verification of Modbus Target

The translation function shall verify the target exists, and shall also attempt to verify that the requested input (T->O) data size (if specified) is valid for the target device. The verification process is accomplished with the following procedure:

1. If the connection request specifies a T->O connection path, the translation function will issue a read request based on the T->O application path using the number of data items indicated in the T->O connection path size. If the T->O application path is the Input Register table, the read request used is the Read Input Register (Function Code 0x04). If the T->O application path is the Holding Register table, the read request used is Read Holding Registers (Function Code 0x03).

   In either case, if the target returns an exception code of Illegal Function (01), the translation function shall return an error with a General Status of 0x01 and an Additional Status of 0x0117 (Invalid Produced or Consumed Application Path). If a returned exception code of Illegal Data Address (02), or Illegal Data Value (03) is received, the translation function shall return an error with a General Status of 0x01 and an Additional Status of 0x0109 (Invalid Connection Size).
2. If the connection request does not specify a T->O connection path, the translation function issues a Read Input Registers request (Function Code 0x04) for one data item starting at Input Register 1. Exception codes of Illegal Function (01), Illegal Data Address (02), or Illegal Data Value (03) are ignored. This process will only detect the nonexistence of a target Modbus device. If the target node is on Modbus/TCP, the translation function can skip this step since the underlying TCP connection management will detect the presence or absence of the target Modbus device.
3. Any returned exception code not listed in 1) and 2) above will result in an error response. The codes returned are listed in the table below.

Modbus Exception Code	CIP General Status Code	Additional Status
Server Device Failure (0x4)	Device State Conflict (0x10)	None
All other codes	Unknown Modbus Error (0x2B)	The actual Modbus exception code received by the translation function is returned as a single 16 bit Additional Status code.

4. If no response is received from the requests issued in 1) and 2) above, an error is returned with a General Status code of 0x01 and an Additional Status code of 0x0204 (Unconnected Request Timed Out).

10-4.2.7 Step 7 – Translation function Returns Forward_Open Response to the Originating Device

The CIP translation function then delivers the Forward_Open Response to the Originating Device. If successful this response will contain the actual packet rates and the connection identifiers to be used. If the verification process failed, an error shall be returned.

10-4.2.7.1 Successful Forward_Open Response

The service data field for a Forward_Open response is shown below.

Table 10-4.2 Forward_Open Response Service Data

Byte Value	Meaning
01 00 00 00	O_to_T Connection ID – In this example, chosen by target node
00 00 00 00	T_to_O Connection ID – In this example, chosen by originating node and echoed back by the target node
0C 00	Connection Serial Number
FF FF	Originator Vendor ID (Vendor ID = 65535)
00 01 02 03	Originator Serial Number (Serial Number = 0x03020100)
00	Connection Timeout Multiplier
00 00 00	Reserved
20 A1 07 00	O_to_T API (0x7A120 = 500 ms)
20 A1 07 00	T_to_O API (0x7A120 = 500 ms)
00	Application reply size (no application reply data)
00	Reserved

10-4.2.7.2 Unsuccessful Forward_Open Response

If any of the verification steps or timing calculations results in an error condition, the translation function shall return an unsuccessful Forward_Open Response.

10-4.2.8 Step 8 – I/O Communication

Once the ForwardOpen response is sent, the connection is active. The I/O timers are running (initially using a preconsumption timer of 10 seconds). Since, in this example, the O_to_T and T_to_O RPI values are the same and both target application paths are assembly objects representing the Holding Register table, the translation function has two choices for communicating on Modbus, as described below.

10-4.2.8.1 I/O Communication with Modbus Target Device Using Read/Write Multiple

When the O_to_T and T_to_O RPI values are the same, and both target application paths are specified (with non-zero connection sizes) to assembly objects representing the Holding Register table, the translation function shall attempt to use the Read/Write Multiple Registers Modbus function (Function Code 0x23) when communicating with the target device. In this case, the translation function does not communicate with the target until the first I/O production is received from the originator. When the first output message arrives, the translation function issues the first Read/Write Multiple Registers function to the target then starts the I/O connection timer based on the API. The data returned from the target is forwarded on to the originator. This function request to the Modbus target device is shown below.

Table 10-4.3 Function Request to Modbus Target Device

Bytes	Meaning
17	Read/Write Multiple Registers function code
13 00	Read starting address (Input Register 20)
05 00	Quantity to read (5 registers)
09 00	Write starting address (Holding Register 10)
04 00	Quantity to write (4 registers)
08	Write byte count (8 bytes)
aa aa bb bb cc cc dd dd	Data to write for Holding Registers 10 - 13

The response from the Modbus target device:

Table 10-4.4 Function Response from Modbus Target Device

Bytes	Meaning
17	Read/Write Multiple Registers function code
0A	Read byte count (10 bytes)
aa aa bb bb cc cc dd dd ee ee	Data read from Holding Registers 20 – 24

The translation function shall issue this request upon each consumption of data from the originator, and the inputs received from the Modbus response are returned to the originator. A production trigger timer is not maintained by the translation function. If output messages from the originator stop, the translation function also stops requesting inputs from the Modbus target device.

If the target device returns an error to the Read/Write Multiple Registers function request, the translation function shall attempt to use the Read Multiple and Write Multiple function pair described in the next section.

10-4.2.8.2 I/O Communication with Modbus Target Device Using Individual Read and Write Functions

When the O_to_T and T_to_O RPI values are not the same, or if either of the connection paths does not specify the Holding Registers, the translation function shall issue individual Read and Write function requests. The write requests are sent when data is received by the translation function from the originator. The read requests are sent upon each expiration of the production trigger timer.

Consider the following example. An Originating Device makes an I/O connection to read Input Register 1, and write Holding Register 2. The table below presents the contents of the Service Data field for the Forward_Open request from the originator to the translation function for this example.

Table 10-4.5 Forward_Open Request Service Data

Byte Value	Meaning
pp	Priority/Tick Time field.
tt	Connection Timeout Ticks field.
FF FF FF FF	O_to_T Connection ID– In this example, chosen by target node1
00 00 00 00	T_to_O Connection ID – In this example, chosen by originating node1
0D 00	Connection Serial Number
FF FF	Originator Vendor ID (Vendor ID = 65535)
00 01 02 03	Originator Serial Number (Serial Number = 0x03020100)
00	Connection Timeout Multiplier
00 00 00	Reserved
20 A1 07 00	O_to_T RPI (0x7A120 = 500 ms)
44 06	O_to_T Connection Parameters2 This parameter value indicates that the consumed connection size is 6 bytes3, is fixed size, High Priority, and Point to Point.
20 A1 07 00	T_to_O RPI (0x7A120 = 500 ms)
44 02	T_to_O Connection Parameters2 This parameter value indicates that the produced connection size is 2 bytes, is fixed size, High Priority, and Point to Point.
01	Transport Type / Trigger (Class 1 Cyclic Client)
10	Connection_Path_Size field indicates that there are 16 words in the connection path.
13 0F 31 33 30 2E 31 35 31 2E 31 33 37 2E 31 30 35 00 20 04 25 00 02 00 20 04 26 00 01 00 01 00	Contents of the Connection_Path field. This field specifies the routing/connection information. These bytes indicate that the request is to be delivered out Port #3 and to IP Address 130.151.137.105 on Modbus/TCP. This path encoding uses the Extended Link Address format for the IP address. It further indicates the applications being connected to by specifying 8 bit Class ID = 4, 16 bit Instance ID = 0x0002 (Holding Register 2) for consumption (O to T), and 8 bit Class ID = 4, 32 bit Instance ID = 0x10001 (Input Register 1) for production (T to O).

1 The allocation of the Connection ID between target and originator is determined by the particular CIP network requirements

2 When connecting to a Word Addressable table the connection size parameter must be a multiple of 2. If not, a General Status code of 0x01 and an Additional Status code of 0x0109 (Invalid Connection Size) shall be returned.

3 The connection size includes 4 bytes for the Real Time Format header.

Data from the originator will contain 1 word of data. This data, when received, is sent to the Modbus target device using function code Write Multiple Registers as shown in the table below.

Table 10-4.6 Write Multiple Registers Function

Bytes	Meaning
10	Write Multiple Registers function code
01 00	Starting address (Holding Register 2)
01 00	Quantity of Registers (1 register)
02	Byte Count (2 byte)
aa bb	Data to write for Holding Register 1

The translation function shall run a production trigger timer based on the connection API. When the timer expires, the inputs are read from the Modbus target device and the timer is restarted. The table below shows the Read Input Registers function that is sent to the Modbus target device.

Table 10-4.7 Read Input Registers Function Request

Bytes	Meaning
04	Read Input Registers function code
00 00	Starting address (Input Register 1)
01 00	Quantity of Input Registers (1 Input register)

Response:

Table 10-4.8 Read Input Registers Function Response

Bytes	Meaning
04	Read Discrete Inputs function code
02	Byte Count (2 byte)
aa bb	Data read for Input register 1

10-4.2.9 Activity Diagrams for I/O Messaging

10-4.2.9.1 Communication Activity Diagram for I/O Produce/Consume

The following activity diagram describes the communication management inside the translation function for a CIP connection that supports both the O_to T and T_to_O directions.

There can only be a maximum of one outstanding request on the Modbus Serial link.

A maximum of one outstanding output request per CIP connection shall be sent to the target Modbus/TCP device.

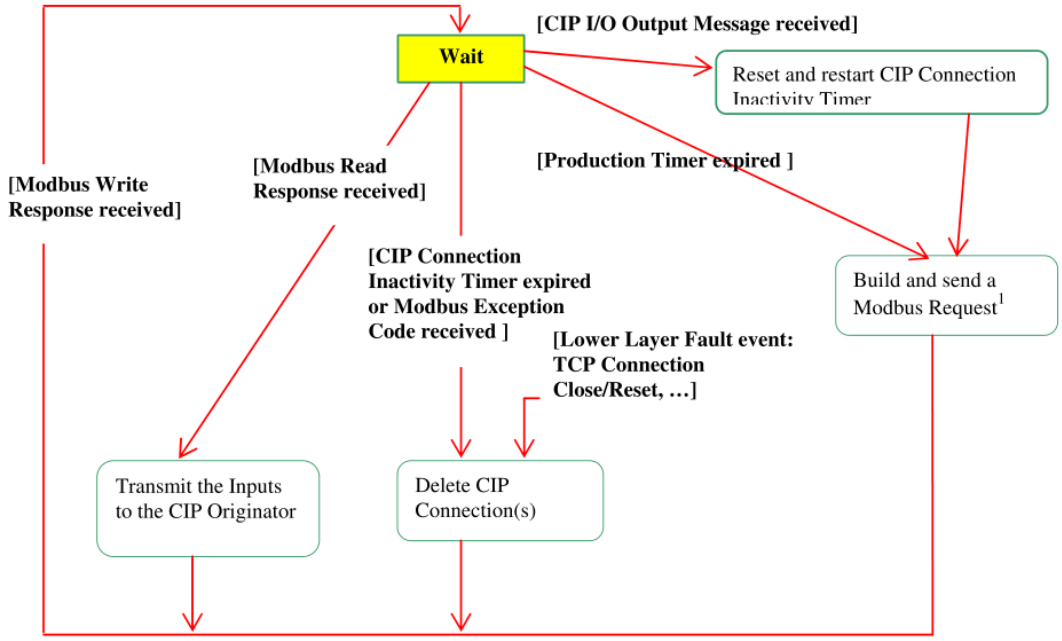
For the T_to_O direction, if the RPI timer expires while there is still an outstanding request, the Modbus Read request will not be sent to the Modbus device. Thus no inputs will be sent back to the CIP Originator and the CIP Inactivity Timer in the CIP Originator may expire. If new output values are received while waiting for the Modbus response from the target device, the translation function shall keep the last received output value (overwriting previous values). This newest value is sent in the next output message (once the Modbus response has been received).

Since the translation function is configured as an originator to target Transport Class 1 connection for the outputs, it shall manage a CIP Inactivity/Watchdog Timer for the connections established between itself and the originating device. In case of an expiration of the CIP Inactivity/Watchdog Timer the translation function shall delete the CIP connection.

In case of no Modbus reply or a Modbus Exception Code returned by the target Modbus device the translation function shall not send any inputs to the CIP Originator. This will allow the CIP originator to detect the expiration of its CIP Inactivity/Watchdog Timer.

On Modbus/TCP, in case of no Modbus reply the TCP Retransmission and Timeout algorithms are activated. Retransmission mechanisms will cause the reset of the TCP connection after an expiration time that may be long (depends on the characteristic of the TCP/IP stack).

Figure 10-4.3 Communication Activity Diagram for I/O Produce/Consume



1 - If a previous send is in progress, store and hold the most recent CIP data for transmission when the send in progress completes.

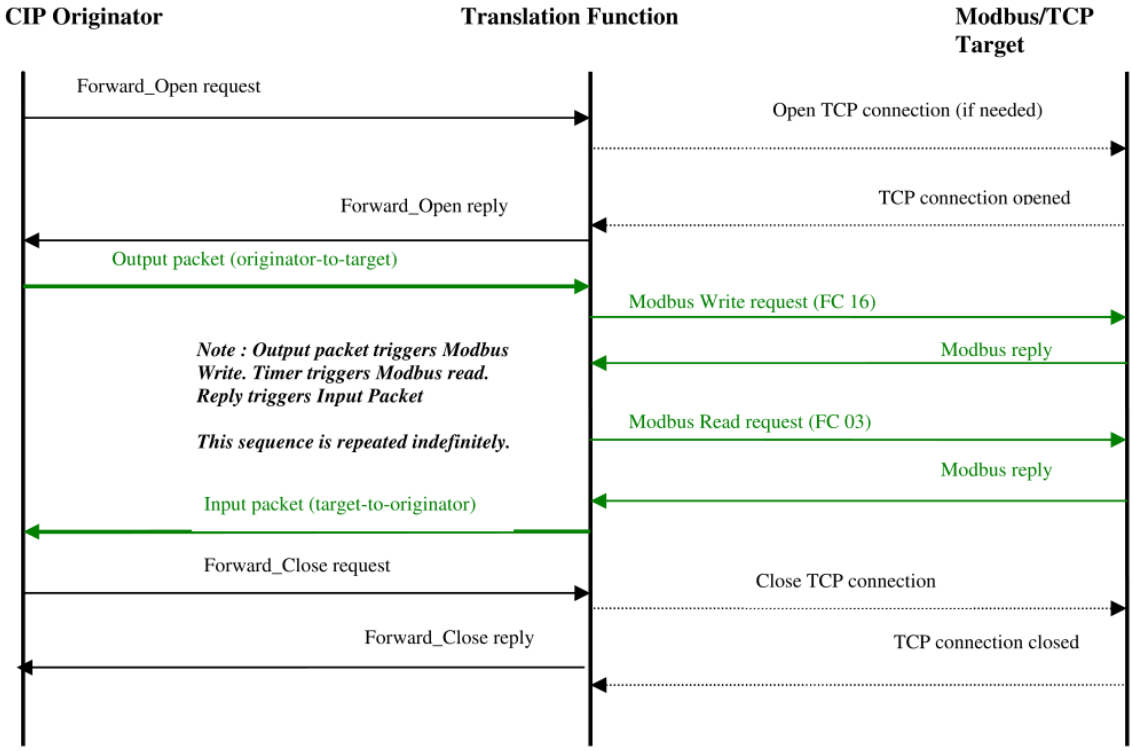
10-4.2.10 Sequence Diagrams for I/O Messaging

This chapter provides some examples of Sequence Diagrams between the different entities of the Modbus/TCP System. The goal is to provide a graphical view of the exchanges in order to offer a better understanding. Basic scenarios without errors and alternatives scenarios of error cases are provided.

10-4.2.10.1 I/O Messaging With No Error (CIP to Modbus/TCP)

The following Sequence Diagram provides an example of I/O communication using Read and Write function requests. Note that in some cases FC 23 will be used instead of individual Read and Write requests.

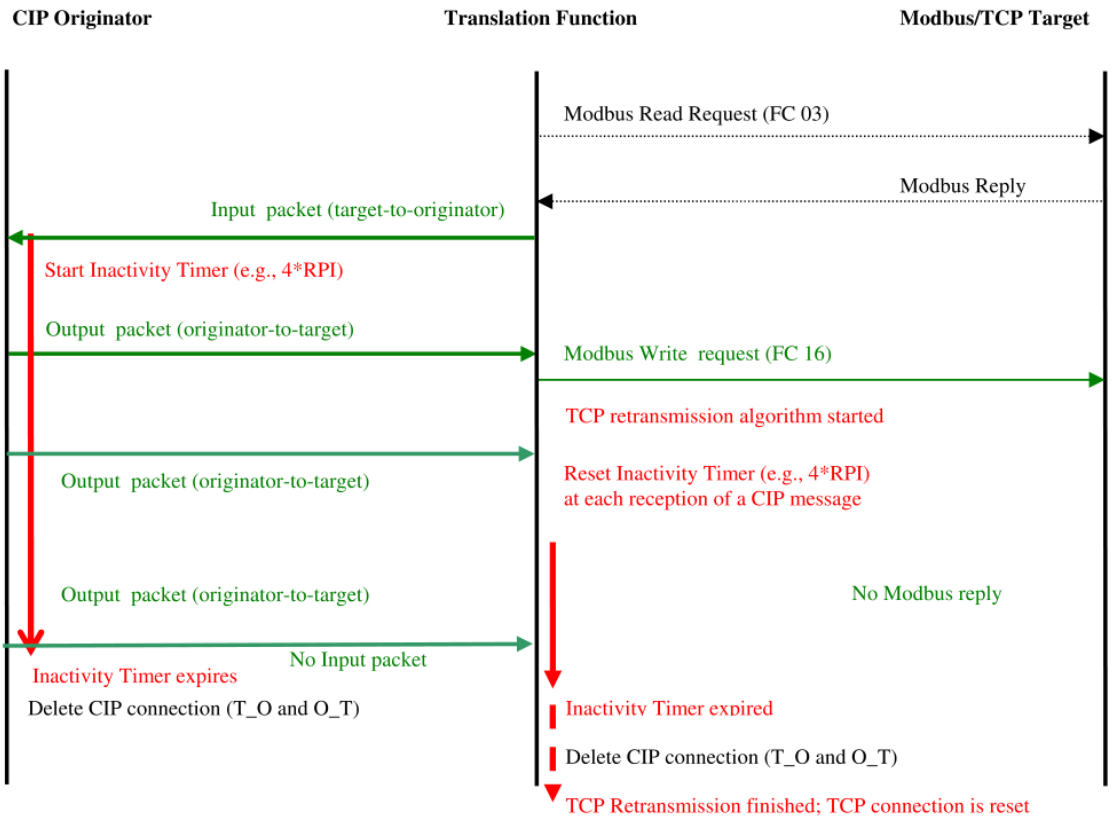
Figure 10-4.4 I/O Message with No Error



10-4.2.10.2 I/O Messaging with Error For O_to_T and T_to_O (No Modbus reply)

The following Sequence Diagrams provide an example of CIP to Modbus/TCP Output/Input Communication with no Modbus reply. In this example it is assumed that the CIP originator's Inactivity Timeout (Timeout Multiplier times RPI) is shorter than the TCP connection timeout.

Figure 10-4.5 I/O Message with Error

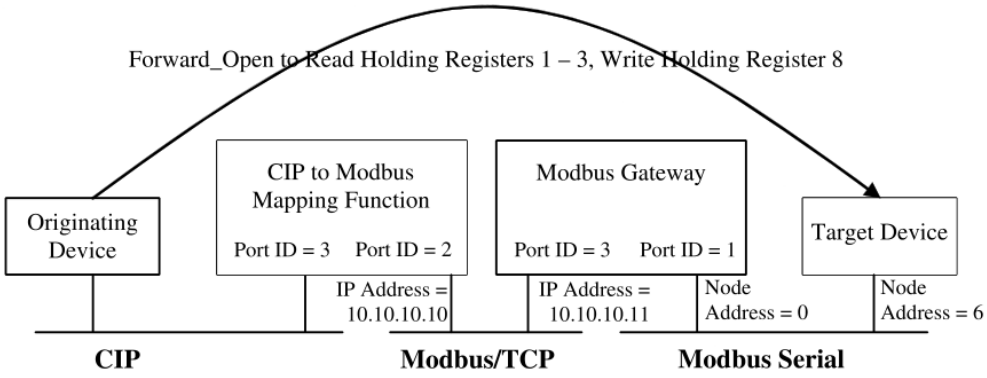


10-4.3 Connected I/O Messaging – Additional Modbus Hop

Connected I/O Messaging is established using the ForwardOpen service of the Connection Manager object. The translation function acts as the proxy for the Modbus target device by processing and responding to the connection request, and translating the subsequent CIP connection data to Modbus function requests.

Assume that the *Originating Device* wants to make an I/O connection to read Holding Registers 1 - 3, and write Holding Register 8.

Figure 10-4.6 Forward_Open to Establish I/O Connection



10-4.3.1 Step 1 – Request delivered to the CIP to Modbus TCP/IP translation function

The first step in the process calls for the Originating Device to issue a Forward_Open request. This example uses two transport class 1 connections; one for I/O production and one for I/O consumption. The table below presents the contents of the Service Data field for the Forward_Open request which is used to execute the connection establishment example noted above (all values are in hex).

Table 10-4.9 Forward_Open Request Service Data

Byte Value	Meaning
pp	Priority/Tick Time field.
tt	Connection Timeout Ticks field.
FF FF FF FF	O_to_T Connection ID– In this example, chosen by target node[3]
00 00 00 00	T_to_O Connection ID – In this example, chosen by originating node[3]
0C 00	Connection Serial Number
FF FF	Originator Vendor ID (Vendor ID = 65535)
00 01 02 03	Originator Serial Number (Serial Number = 0x03020100)
00	Connection Timeout Multiplier
00 00 00	Reserved
20 A1 07 00	O_to_T RPI (0x7A120 = 500 ms)
44 06	O_to_T Connection Parameters[1] This parameter value indicates that the consumed connection size is 2 bytes[4], is fixed size, High Priority, and Point to Point.
20 A1 07 00	T_to_O RPI (0x7A120 = 500 ms)
44 08	T_to_O Connection Parameters[1] This parameter value indicates that the produced connection size is 8 bytes, is fixed size, High Priority, and Point to Point.
01	Transport Type / Trigger (Class 1 Cyclic Client)
10	Connection_Path_Size field indicates that there are 16 words in the connection path.
12 0B 31 30 2E 31 30 2E 31 30 2E 31 31 00 01 06 21 00 04 00 2D 00 08 00 21 00 04 00 2D 00 01 00	Contents of the Connection_Path field. This field specifies the routing/connection information. These bytes indicate that the request is to be delivered out Port #2 and to IP Address 10.10.10.11 on Modbus/TCP; then out Port #1 to MAC ID 6. It further indicates the applications being connected to by specifying 16 bit Class ID = 4, 16 bit Connection Point[2] = 8 for consumption (O to T), and 16 bit Class ID = 4, 16 bit Connection Point[2] = 1 for production (T to O). Note that for the Assembly Object, the attribute is assumed to be the Data attribute (Attribute ID 3) if not present in the EPATH.

Byte Value	Meaning
1	The connection size parameter must be a multiple of 2. If not, a General Status code of 0x01 and an Additional Status code of 0x0109 (Invalid Connection Size) shall be returned.
2	The Assembly class has defined all connection points to be equivalent to the instance number. See Volume 1, Chapter 5.
3	The allocation of the Connection ID between target and originator is determined by the particular CIP network requirements.
4	The connection size includes 4 bytes for the Real Time Format header.

The Forward_Open request has now been delivered to the CIP/Modbus translation function.

10-4.3.2 Step 2 – Translation function creates a CIP connection with the Originator

Upon receipt of a Forward_Open to the Connection Manager object, the translation function shall (if it has the resources available) create a CIP connection between itself and the originating device. If resources are unavailable, an error response is returned.

10-4.3.3 Step 3 – Translation function creates virtual connection with the Target Modbus Device

After the Connection Manager Object within the CIP translation function processes the Forward_Open request it examines the contents of the Connection_Path field. In this case, the contents are "12 0B 31 30 2E 31 30 2E 31 30 2E 31 31 00 01 06 21 00 04 00 2D 00 08 00 21 00 04 00 2D 00 01 00". This indicates that the next device in the hop is on Port #2 and its Node (IP) Address is 10.10.10.11 on Modbus/TCP. Additionally, since there is one more Port/Node Address pair specified in the Connection_Path field, the request has not reached its destination network. The next hop, specified by "01 06" indicates the target address. The CIP/Modbus translation function shall only support an additional hop through Port #1 (the backplane port). The address specified in this last hop (node address 6 in this example) is placed into the Modbus/TCP Unit ID field.

The translation function shall exhibit the same behavior as the single hop case.

10-4.4 Explicit Messaging

The CIP to Modbus translation function translates CIP explicit message requests and responses to Modbus function requests and responses. Only those service/object translations defined by this specification shall be translated by the translation function. Valid objects for explicit message requests are:

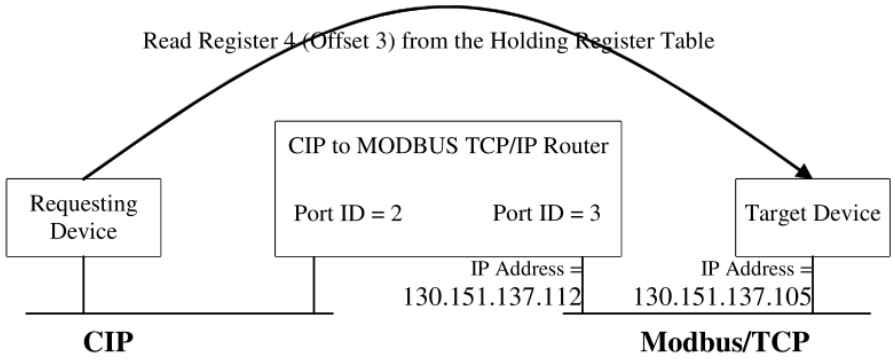
- Identity Object
- Assembly Object
- Parameter Object
- Modbus Object

10-4.4.1 Unconnected Explicit Messaging

Unconnected explicit messaging uses the Unconnected_Send service of the Connection Manager object to deliver an explicit messaging request from a client to a server without using an already established connection.

Assume that the *Requesting Device* wants to read a holding register (register 4 at offset 3 in the Modbus Holding Register table). This is accomplished with a Get_Attribute_Single to the Parameter Object (Class #15/Instance #4/Attribute ID #1) in the *Target Device*.

Figure 10-4.7 Unconnected Send (Explicit Message Request/Response)



10-4.4.1.1 Step 1 – Request delivered to the CIP to Modbus TCP/IP translation function

The first step in the process calls for the Requesting Device to issue the Unconnected Send request.

The table below presents the contents of the Service Data field for the Unconnected Send request which is used to execute the Read (Get_Attribute_Single) example noted above (all values are in hex).

Table 10-4.10 Unconnected Send Request Service Data

Bytes	Meaning
pp	Priority/Tick Time field.
tt	Connection Timeout Ticks field.
0A 00	Message_Size field = 10 bytes.
0E	Service field = Get_Attribute_Single.
04	Request_Path_Size field indicates that there are 4 words in the path field.
20 0F 25 00 04 00 30 01	Path field specifying 8 bit Class Id = 15, 16 bit Instance ID = 4, 8 bit Attribute Id = 1. Note the presence of the pad bytes immediately following the 0x25 byte. This path could have also been formatted as follows: 20 0F 24 04 30 01 which makes use of an 8 bit value to convey the Instance ID. In the 8 bit case, the pad byte is not required.
09	Route_Path_Size field indicates that there are 9 words in the route path.
00	Reserved byte.
13 0F 31 33 30 2E 31 35 31 2E 31 33 37 2E 31 30 35 00	Contents of the Route_Path field. This field specifies the routing information. These bytes indicate that the request is to be delivered out Port #3 and to IP Address 130.151.137.105 on Modbus/TCP. This path encoding uses the Extended Link Address format for the IP address.

10-4.4.1.2 Step 2 – Translation function delivers the request to the Target Device

When the Connection Manager Object within CIP/Modbus translation function receives the Unconnected Send request it examines the contents of the *Route_Path* field. In this case, the contents are "13 0F 31 33 30 2E 31 35 31 2E 31 33 37 2E 31 30 35 00". This indicates that the next device in the hop is on Port #3 and its Node (IP) Address is 130.151.137.105 on Modbus/TCP. Additionally, since there is only a single Port/Node Address pair specified in the *Route_Path*, the request has reached its destination network. When a CIP/Modbus translation function receives an Unconnected Send Request that DOES NOT need to be forwarded through more intermediate routers, the following steps are taken:

- The translation function executes the timing related logic associated with the *Priority/Tick Time* and *Connection Timeout Ticks* fields. If a Timeout is detected, then a successful Unconnected Send response that specifies a Routing Error is returned to the Requesting Device.
- The CIP translation function extracts the actual transaction from the Message Request portion of the Service Data, translates the CIP request into a Modbus function request, and delivers the message to the Target Device. The actual method of delivering the explicit message is dependant on the Modbus link type.

In this example, the message request inside the Unconnected Send indicates a Get_Attribute_Single to Class #15, Instance #4, Attribute #1. This translates into a Modbus Read Holding Registers request. The format of the Modbus request is shown in the table below. The data is represented in hex and, when multiple bytes are shown in a cell, in little endian format.

Table 10-4.11 Modbus Read Holding Registers Request

Bytes	Meaning
03	Read Holding Registers function code
03 00	Starting address for holding register 4
01 00	Quantity of registers is 1

10-4.4.1.3 Step 3 – Target returns the Explicit Message response to the translation function

The Target Device processes the Modbus function and returns a response to the translation function. The response is shown in the table below.

Table 10-4.12 Modbus Read Holding Registers Response

Bytes	Meaning
03	Read Holding Registers function code
02	Byte count = 2 bytes
xx yy	Register value

When the response is received by the translation function it translates the Modbus function response into the proper CIP response.

10-4.4.1.4 Step 4 – Translation function returns the Unconnected Response to the Requesting Device

The translation function then delivers the Unconnected Send Response to the original Requesting Device. If successful, Table 10-4.13 indicates the response returned by the translation function on the upstream CIP network.

Table 10-4.13 Unconnected Send Response Service Data

Bytes	Meaning
D2	Reply service code
00	Reserved
00	General Status
00	Reserved
xx yy	Service response data

If an error was returned on Modbus, see section 10-4.4.3 for the appropriate translation of Modbus error conditions to CIP error codes.

10-4.4.2 Connected Explicit Messaging

A Connected Explicit messaging connection is established using the ForwardOpen service of the Connection Manager object, with an application path of Class Message Router, Instance 1. See Volume 1, Chapter 3 for additional details about establishing CIP Explicit Messaging Connections.

The following table shows the data field values used in a connection request for an Explicit Messaging connection.

Table 10-4.14 Forward_Open Request Service Data

Byte Value	Meaning
pp	Priority/Tick Time field.
tt	Connection Timeout Ticks field.
00 00 00 00	O_to_T Connection ID – In this example, chosen by originating node[1]
FF FF FF FF	T_to_O Connection ID – In this example, chosen by target node[1]
0C 00	Connection Serial Number
FF FF	Originator Vendor ID (Vendor ID = 65535)
00 01 02 03	Originator Serial Number (Serial Number = 0x03020100)
00	Connection Timeout Multiplier
00 00 00	Reserved
A0 25 26 00	O_to_T RPI (0x2625A0 = 2500 ms)
46 80	O_to_T Connection Parameters. This parameter value indicates that the maximum consumed connection size is 128 bytes, is variable size, High Priority, and Point to Point.
A0 25 26 00	T_to_O RPI (0x2625A0 = 2500 ms)
46 80	T_to_O Connection Parameters. This parameter value indicates that the maximum consumed connection size is 128 bytes, is variable size, High Priority, and Point to Point.
83	Transport Type / Trigger (Class 3 Server, production trigger ignored by target)
09	Connection_Path_Size field indicates that there are 9 words in the connection path.

Byte Value	Meaning
12 0B 31 30 2E 31 30 2E 31 30 2E 31 31 00 20 02 24 01	Contents of the Connection_Path field. This field specifies the routing/connection information. These bytes indicate that the request is to be delivered out Port #2 and to IP Address 10.10.10.11 on Modbus/TCP. It further indicates the applications being connected to by specifying 8 bit Class ID = 2 (Message Router) and 8 bit Instance = 1.

1 The allocation of the Connection ID between target and originator is determined by the particular CIP network requirements.

Once established, data flowing across this established connection will conform to the Message Router Request and Response formats as specified in Volume 1, Chapter 2. The translation function shall examine all requests, and only translate those objects as defined earlier in this section. An error response of General Status 0x16, with no Additional Status, is returned to a request to for any other object.

The translation function is not responsible for detecting CIP Explicit Messaging retry messages. The translation function shall ignore the sequence count and forward all explicit message requests to the target Modbus device. On Modbus/TCP, there is no correlation between the CIP sequence count and the Modbus/TCP Transaction Identifier. However, on all Modbus networks, the translation function shall match CIP requests (both connected and unconnected) to the associated Modbus responses.

10-4.4.3 Explicit Messaging Error Translation

The following table provides the translation between Modbus Exception Codes and CIP General Status Codes.

Table 10-4.15 Modbus Exception Code to CIP General Status Code Translation

Modbus Exception Code	CIP General Status Code	Additional Status
Illegal Function Code (0x01)	Service Not Supported (0x08)	None
Illegal Data Address (0x02)	Object Does Not Exist (0x16)	None
Illegal Data Value (0x03)	Invalid Parameter Value (0x03)	None
Server Device Failure (0x04)	Device State Conflict (0x10)	None
Server Device Busy (0x06)	Resource Unavailable (0x02)	None
Gateway Path Unavailable (0x0A)	Connection failure (0x01)	Link Address Not Valid (0x0312)
Target Device Failed to Respond (0x0B)	Connection failure (0x01)	Unconnected Request Timed Out (0x0204)
All other codes	Unknown Modbus Error (0x2B)	The actual Modbus exception code received by the translation function is returned as a single 16 bit Additional Status code.

10-4.4.4 Activity Diagrams for Explicit Messaging

10-4.4.4.1 Unconnected Explicit Messaging Activity Diagram

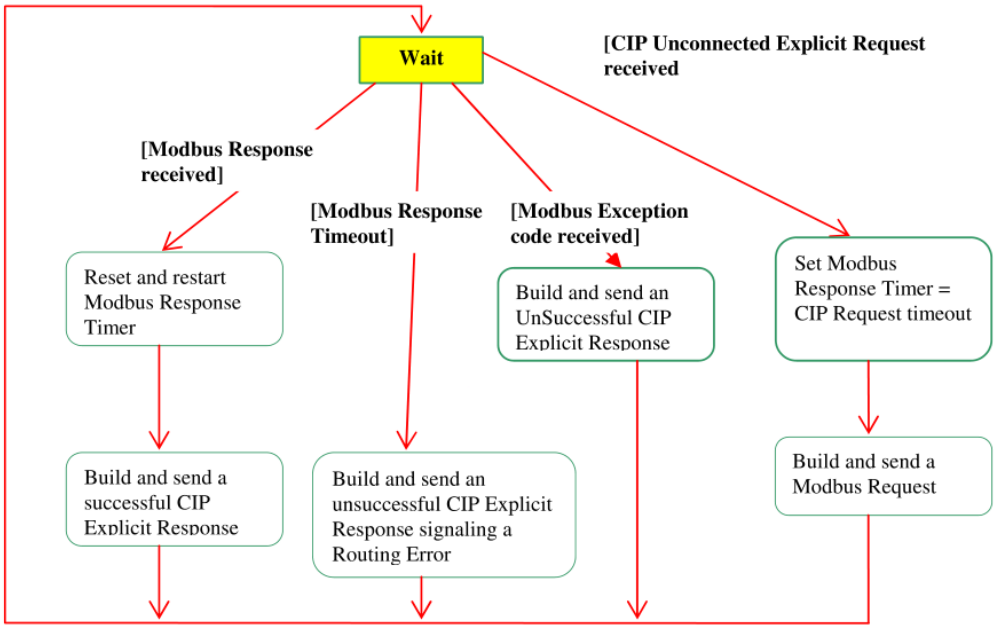
The following activity diagram describes the Unconnected Explicit Messaging management inside the translation function.

Several outstanding Unconnected Explicit Messages may be received by the translation function for the same target Modbus device. If the translation function does not have the capability to handle them, it shall return an unsuccessful CIP Explicit Response that specifies a Resource Unavailable Error (General Status 0x02). If the target Modbus device does not have the capability to handle a request, it may return a Modbus Exception Response Server Device Busy Exception Code(0x06), which the translation function translates into an unsuccessful CIP Explicit Response that specifies a Resource Unavailable Error.

In the case of a Modbus Response with no error received, the translation function shall send a successful Explicit Message Response to the originator. In the case of a Modbus Exception Code received, the translation function shall send an unsuccessful Explicit Message Response to the originator with a General Status code based on the Modbus Exception code (see 10-4.4.3).

The Modbus Application Layer Timeout mechanisms shall be implemented. The translation function shall set the Modbus Response Timeout value to the CIP request Timeout before sending the Modbus Request. The translation function shall reset the Modbus Response Timer at each reception of a Modbus Response. In case of expiration of the Modbus Response Timer the translation function shall return an unsuccessful CIP Explicit Response that specifies a Routing Error (General Status 0x01, Additional Status 0x0204) to the Originating Device.

Figure 10-4.8 Unconnected Explicit Messaging Activity Diagram



10-4.4.4.2 Connected Explicit Messaging Activity Diagram

The following Activity Diagram describes the Connected Explicit Messaging management inside the translation function.

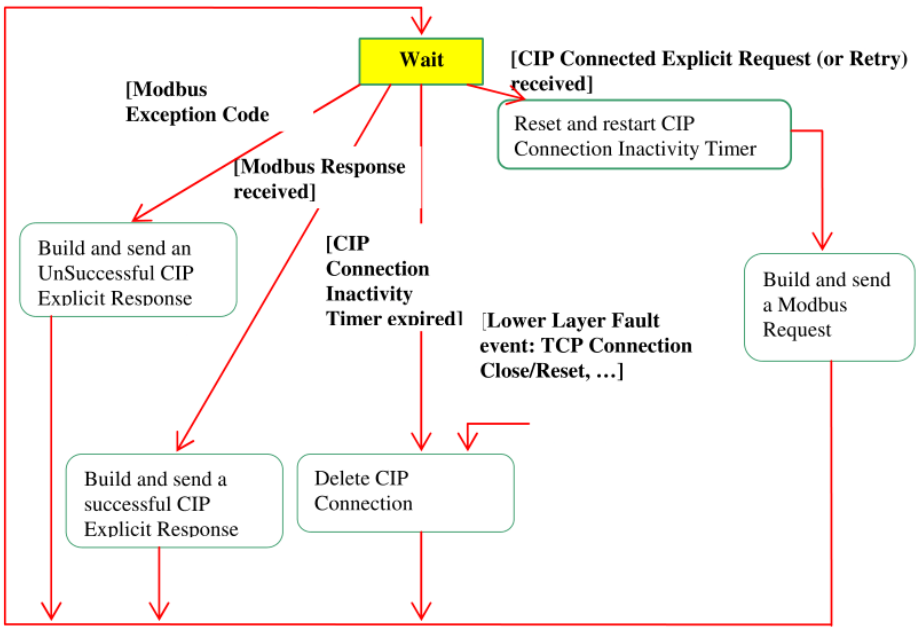
Several outstanding Connected Explicit Messages can be sent by the CIP Originator to the same target Modbus device. If the translation function does not have the capability to handle them, it shall return an unsuccessful CIP Explicit Response that specifies a Resource Unavailable Error (General Status 0x02). If the target Modbus device does not have the capability to handle a request, it may return a Modbus Exception Response Server Device Busy Exception Code (0x06), which the translation function translates into an unsuccessful CIP Explicit Response that specifies a Resource Unavailable Error

For connected Explicit Messages the translation function is responsible for managing the Inactivity/Watchdog Timer for the CIP connection established between itself and the originating device. In case of an expiration of the Inactivity/Watchdog Timer, the translation function shall delete the CIP connection.

In the case of a Modbus Response with no error received, the translation function shall send a successful Explicit Message Response to the originator. In the case of a Modbus Exception Code received, the translation function shall send an unsuccessful Explicit Message Response to the CIP Originator with a General Status code based on the Modbus Exception code (see 10-4.4.3)

In the case of no Modbus reply the translation function shall not send any Explicit Message Response to the originator. This will allow the CIP originator to detect the expiration of its Inactivity/Watchdog Timer. In parallel the TCP retransmission algorithm is activated and will cause the reset of the TCP connection after an expiration time that may be long (depends on the characteristic of the TCP/IP stack). The reset of the TCP connection will cause the delete of the CIP connection of the translation function if it has not already been deleted by the CIP Inactivity/watchdog detection.

Figure 10-4.9 Connected Explicit Messaging Activity Diagram



10-4.4.5 Sequence Diagrams for Explicit Messaging

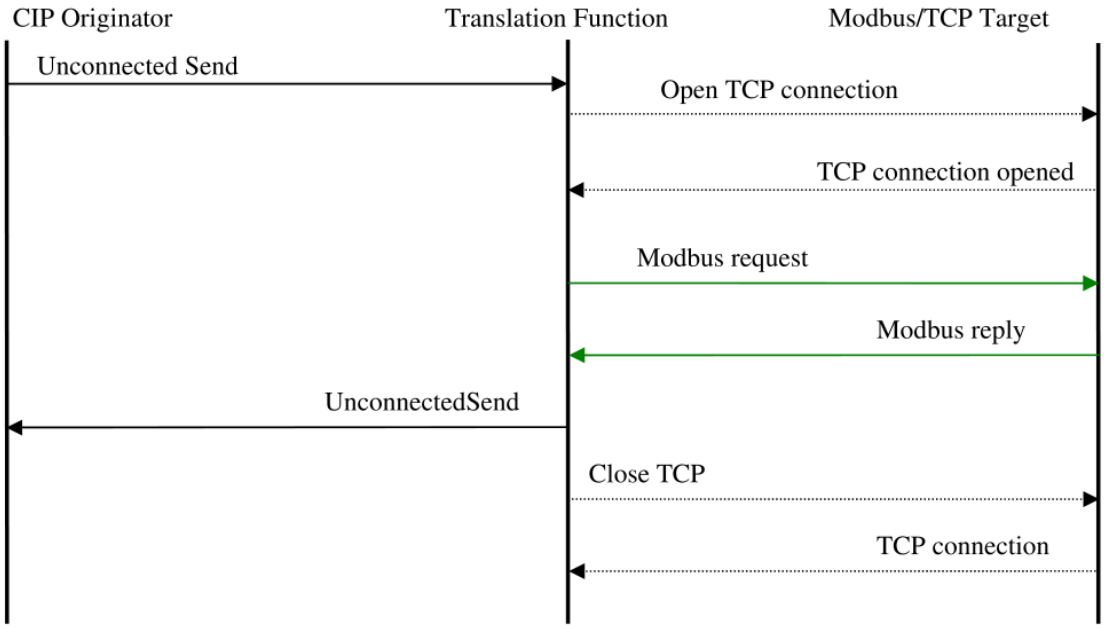
The following sections provide sequence diagrams for both connected and unconnected explicit messaging relating to:

- connection and timing management
- error handling.

10-4.4.5.1 Unconnected Explicit Messaging with No Error

The following Sequence Diagram provides an example of Unconnected Explicit Messaging exchange with no error.

Figure 10-4.10 Unconnected Explicit Message Sequence Diagram – No Error

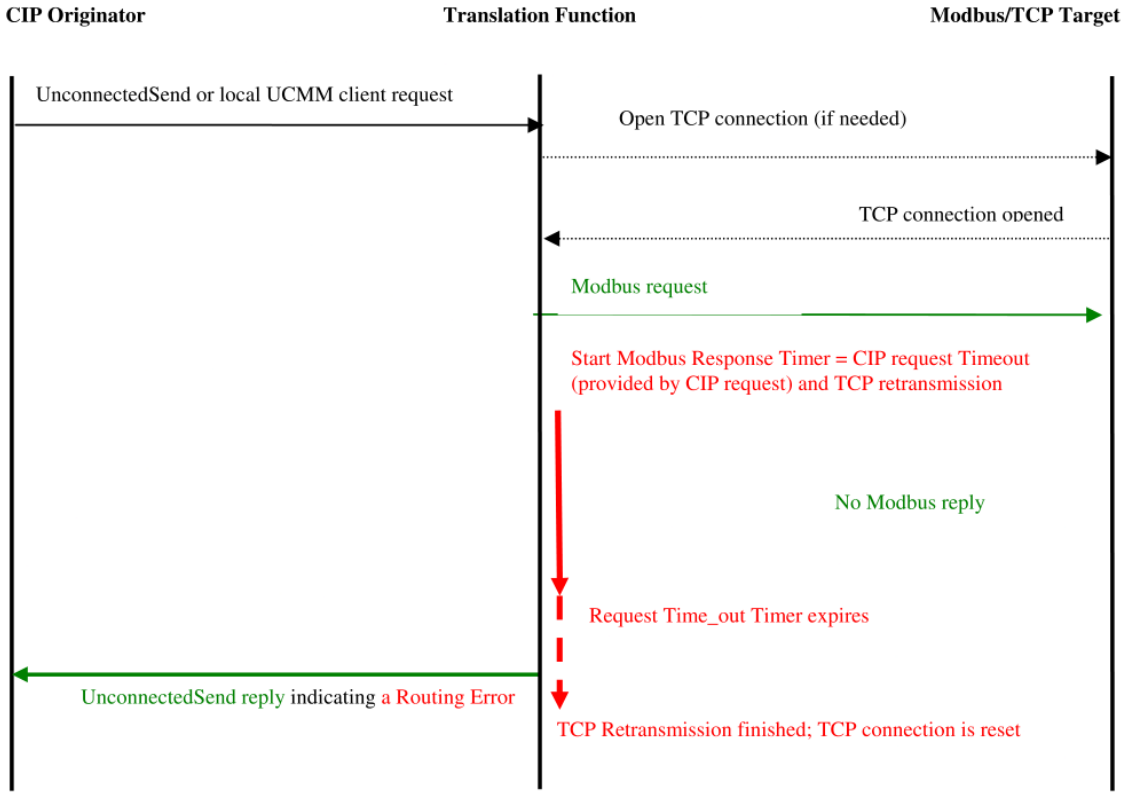


10-4.4.5.2 Unconnected Explicit Messaging with Error (No Modbus reply)

The following Sequence Diagrams provide an example of CIP to Modbus/TCP Unconnected Explicit Messaging Communication with no Modbus reply.

This example assumes that the CIP request timeout is shorter than the TCP connection timeout. An unsuccessful CIP Explicit Response that specifies a Routing Error is sent by the translation function before the reset of the TCP connection (see diagram below). This means that a Modbus response could be received from the Modbus target device after the unsuccessful CIP Explicit Response has been sent. In this case the Modbus response will be dropped by the translation function.

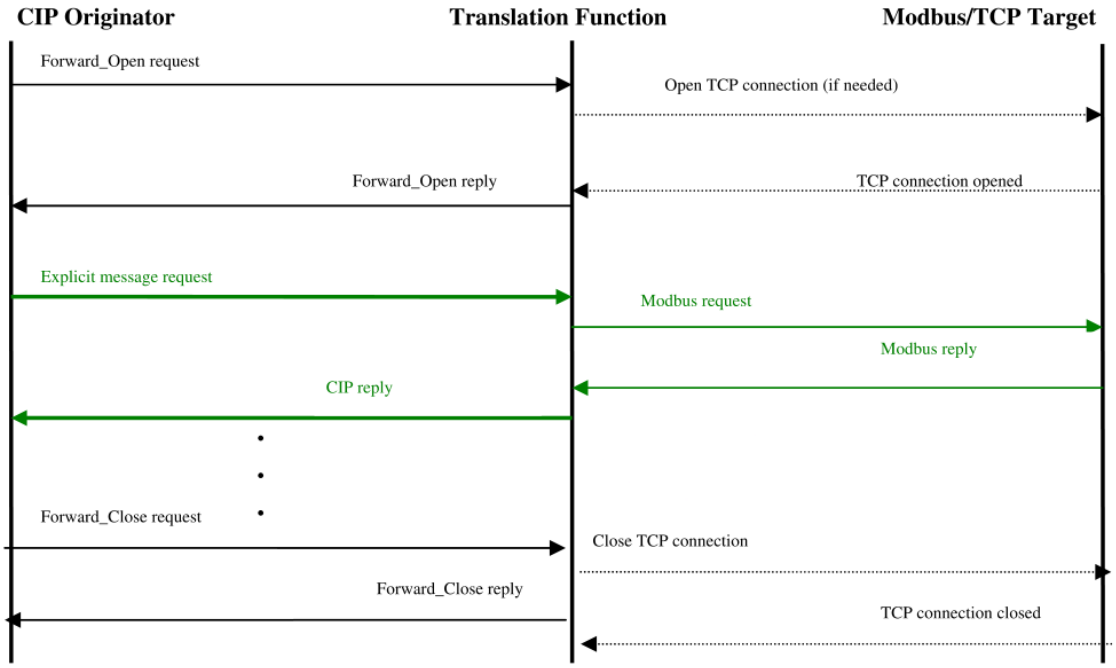
Figure 10-4.11 Unconnected Explicit Message Sequence Diagram – With Error



10-4.4.5.3 Connected Explicit Messaging: Example of Exchange with No Error

The following Sequence Diagram provides an example of Connected Explicit Messaging exchange with no error.

Figure 10-4.12 Connected Explicit Message Sequence Diagram – No Error

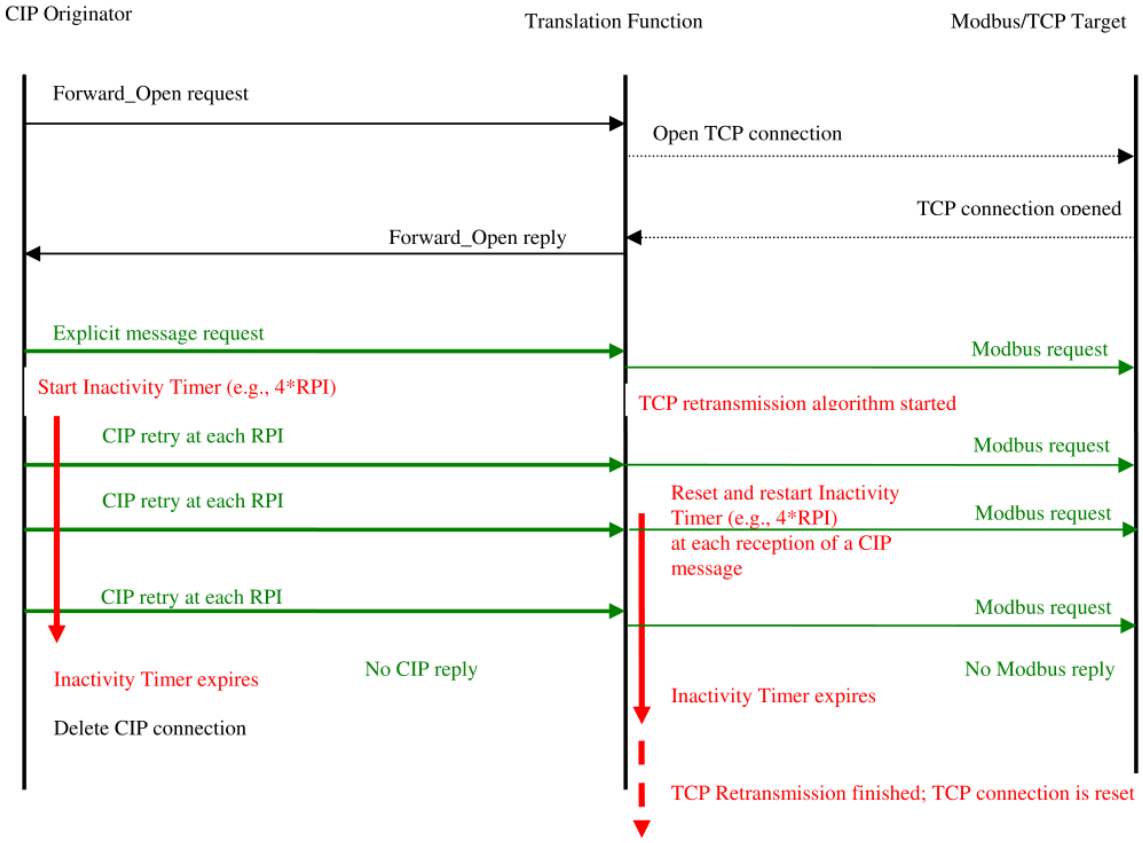


10-4.4.5.4 Connected Explicit Messaging with Error (No Modbus reply)

The following Sequence Diagrams provide an example of CIP to Modbus/TCP connected Explicit Messaging communication with No Modbus reply.

This example assumes that the CIP originator Inactivity Timeout is shorter than the TCP connection timeout. So a Modbus response could be received from the Modbus target device after the CIP connection has been deleted in the translation function. In this case the Modbus response will be dropped by the translation function.

Figure 10-4.13 Connected Explicit Message Sequence Diagram – With Error



10-4.5 Using the Forward_Close Service to Close a Connection

The table below presents the contents of the Service Data field for the Forward_Close request to delete the connection which was established in the example above (all values are in hex).

Table 10-4.16 Forward_Close Request Service Data

Byte Value	Meaning
pp	Priority/Tick Time field.
tt	Connection Timeout Ticks field.
0C 00	Connection Serial Number
FF FF	Originator Vendor ID (Vendor ID = 65535)
00 01 02 03	Originator Serial Number (Serial Number = 0x03020100)
0E	Connection_Path_Size field indicates that there are 14 words in the connection path.
00	Reserved
13 0F 31 33 30 2E 31 35 31 2E 31 33 37 2E 31 30 35 21 00 04 00 25 00 02 00 30 03 00	Contents of the Connection_Path field. This field specifies the routing/connection information. These bytes indicate that the request is to be delivered out Port #3 and to IP Address 130.151.137.105 on Modbus/TCP. This path encoding uses the Extended Link Address format for the IP address. It further indicates the application being disconnected from by specifying 16 bit Class ID = 4, 16 bit Instance ID = 2, 8 bit Attribute ID = 3.

10-4.5.1 Step 1 – Request Delivered to the CIP to Modbus Translation Ffunction

The connection originator sends a Forward_Close request to the Modbus target device. The Connection Manager in the translation function, as a CIP router in the path, receives this request, processes the parameter values, and attempts to find a matching connection.

10-4.5.2 Step 2 – Translation Function Deletes the CIP Bridged Connection with Target

After the Connection Manager Object within the CIP translation function processes the Forward_Close request it examines the contents of the *Connection_Path* field. The data in this field indicates that the next device in the hop is on Port #3 and its Node (IP) Address is 130.151.137.105. Additionally, since there is only a single Port/Node Address pair specified in the *Connection_Path*, the request has reached its destination network. When a CIP translation function receives a Forward_Close Request that DOES NOT need to be forwarded through more intermediate CIP Routers, the following steps are taken:

The CIP translation function executes the timing related logic associated with the *Priority/Tick Time* and *Connection Timeout Ticks* fields. If a Timeout is detected, then a successful Forward_Close response that specifies a Routing Error is returned to the Requesting Device.

10-4.5.3 Step 3 – Translation Function Returns the Forward_Close Response to the Requesting Device and Releases Internal Resources

The CIP translation function then delivers the Forward_Close Response to the Originating Device and releases all internal resources related to the connections.

The service data field for a Forward_Close response is shown below.

Table 10-4.17 Forward_Close Response Service Data

Byte Value	Meaning
0C 00	Connection Serial Number
FF FF	Originator Vendor ID (Vendor ID = 65535)
00 01 02 03	Originator Serial Number (Serial Number = 0x03020100)
00	Application reply size (no application reply data)
00	Reserved

This page intentionally left blank